

CICS Transaction Server for z/OS



CICS Web Services Guide

Version 3 Release 1

CICS Transaction Server for z/OS



CICS Web Services Guide

Version 3 Release 1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 247.

Ninth edition (July 2010)

This edition applies to Version 3 Release 1 of CICS Transaction Server for z/OS, program number 5655-M15, 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.

© Copyright IBM Corporation 2005, 2010.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	ix
What this book is about	ix
Who should read this book.	ix
Chapter 1. CICS and Web Services	1
What is a Web service?	1
How Web services can help your business	2
Web services terminology	2
Chapter 2. The Web services architecture	7
The Web service description	8
Service publication	10
Chapter 3. What is SOAP?	11
The structure of a SOAP message.	11
The SOAP header	13
The SOAP body	15
The SOAP fault.	15
SOAP nodes.	16
The SOAP message path	17
Chapter 4. How CICS supports Web services.	19
Message handlers and pipelines	19
Transport-related handlers.	20
Interrupting the flow	21
A service provider pipeline.	21
A service requester pipeline	22
CICS pipelines and SOAP.	23
SOAP messages and the application data structure	24
WSDL and the application data structure	26
The Web service binding file	28
External standards	29
Extensible Markup Language Version 1.0	29
SOAP 1.1 and 1.2.	29
Web Services Description Language Version 1.1	30
Web Services Coordination Version 1.0	30
Web Services Atomic Transaction Version 1.0	30
WS-I Basic Profile Version 1.1	30
WS-I Simple SOAP Binding Profile Version 1.0	31
Web Services Security: SOAP Message Security	31
Web Services Security: UsernameToken Profile 1.0	32
Web Services Security: X.509 Certificate Token Profile 1.0.	32
XML Encryption Syntax and Processing.	32
XML-Signature Syntax and Processing	32
CICS compliance with Web services standards	33
Chapter 5. Getting started with Web services	37
Planning to use Web services	37
Planning a service provider application	38
Planning a service requester application	40
Migrating from the SOAP for CICS feature.	41
Chapter 6. Configuring your CICS system for Web services	45

CICS resources for Web services	45
Configuring CICS to use the WebSphere MQ transport	47
The WebSphere MQ transport	48
Defining local queues in a service provider	49
Defining local queues in a service requester	50
The URI for the MQ transport	50
Configuring CICS to support persistent messages	51
Persistent message processing	52
Chapter 7. The CICS Web services assistant	53
Using the CICS Web services assistant to deploy a service provider application	54
Creating a service provider application from a Web service description	54
Creating a service provider application from a data structure	55
Customizing generated Web service description documents	56
Using the CICS Web services assistant to deploy a service requester application.	57
Creating a service requester application from a Web service description	57
Creating the CICS infrastructure for a service provider	58
Creating the CICS infrastructure for a service requester	60
Invoking the Web services assistant using an API	61
Validating SOAP messages	61
DFHLS2WS: high level language to WSDL conversion	62
DFHWS2LS: WSDL to high level language conversion	70
Mapping levels for the CICS Web services assistant	80
High level language and XML schema mapping	83
COBOL and XML Schema mapping	84
C/C++ and XML Schema mapping	91
PL/I and XML Schema mapping	97
Variable arrays of elements	104
Support for XML attributes	107
Chapter 8. Interfacing with service provider and requester applications	111
How an application is invoked in a service provider	111
How CICS invokes a service provider program deployed with the Web services assistant.	111
How CICS invokes other service provider programs	112
Invoking a Web service from a CICS program	113
Invoking a Web service from an application deployed with the Web services assistant	113
Invoking a Web service from other applications	114
Run time limitations for code generated by the Web services assistant	115
Chapter 9. The pipeline configuration file	119
Transport-related handlers	120
The pipeline definition for a service provider.	122
The pipeline definition for a service requester	123
Elements used only in service providers	124
The <named_transport_entry> element	125
The <provider_pipeline> element	125
The <terminal_handler> element	126
The <transport_handler_list> element	127
Elements used in service requesters	127
The <requester_pipeline> element.	127
Elements used in service provider and requesters	128
The <cics_soap_1.1_handler> element	128
The <cics_soap_1.2_handler> element	130

The <default_http_transport_handler_list> element	132
The <default_mq_transport_handler_list> element	133
The <default_transport_handler_list> element	133
The <handler> element	134
The <service> element	134
The <service_handler_list> element	135
The <service_parameter_list> element	135
The <transport> element	136
Pipeline configuration for WS-Security	137
The <wsse_handler> element	137
The <dfhwsse_configuration> element	137
The <authentication> element	138
The <sign_body> element	141
The <encrypt_body> element	142
Chapter 10. Message handlers	145
Message handler protocols	145
Supplying your own message handlers	147
Working with messages in a non-terminal message handler	148
Passing a message to the next message handler in the pipeline	149
Forcing a transition to the response phase of the pipeline.	149
Suppressing the response	149
Handling one way messages in a service requester pipeline.	149
Working with messages in a terminal message handler	150
Handling errors	151
The message handler interface	151
Chapter 11. The SOAP message handlers	153
Header processing programs	153
The header processing program interface	155
The SOAP handler interfaces	157
The application interface	157
Dynamic routing of inbound requests in a terminal handler	158
Chapter 12. Containers used in the pipeline.	159
The control containers.	159
Container DFHERROR	159
Container DFHFUNTION	161
Container DFHHTTPSTATUS	163
Container DFHMEDIATYPE.	164
Container DFHNORESPONSE	164
Container DFHREQUEST	164
Container DFHRESPONSE	165
How containers control the pipeline protocols	165
The context containers	168
Container DFH-HANDLERPLIST	168
Container DFH-SERVICEPLIST	168
Container DFHWS-APPHANDLER	168
Container DFHWS-DATA.	169
Container DFHWS-OPERATION	169
Container DFHWS-PIPELINE	169
Container DFHWS-SOAPLEVEL	169
Container DFHWS-TRANID.	170
Container DFHWS-URI	170
Container DFHWS-USERID.	170
Container DFHWS-WEBSERVICE	170

User containers	170
Chapter 13. Support for Web Services transactions.	171
Registration services	171
Configuring CICS for Web service transactions	173
Configuring a service provider for Web service transactions	174
Configuring a service requester for Web service transactions	175
Determining if the SOAP message is part of an atomic transaction	177
Checking the progress of an atomic transaction	178
Chapter 14. Support for Web Services Security	179
Prerequisites	179
The options for securing SOAP messages	180
Signing of SOAP messages.	181
Signature algorithms	182
Example of a signed SOAP message	182
CICS support for encrypted SOAP messages	183
Encryption algorithms	184
Example of an encrypted SOAP message	184
Configuring RACF for Web Services Security	185
Configuring CICS for Web Services Security	187
Chapter 15. Diagnosing problems.	191
Diagnosing deployment errors	191
Diagnosing service provider runtime errors	192
Diagnosing service requester runtime errors.	193
Diagnosing data conversion errors	195
Why data conversion errors occur	195
Conversion errors in trace points	196
SOAP fault messages for conversion errors	198
Chapter 16. The CICS catalog manager example application	199
The base application	199
BMS presentation manager	201
Data handler	201
Dispatch manager	201
Order dispatch endpoint	201
Stock manager	201
Application configuration	202
Installing and setting up the base application	202
Creating and defining the VSAM data sets	202
Defining the 3270 interface	203
Completing the installation	204
Configuring the example application.	204
Running the example application with the BMS interface	206
Web service support for the example application	208
Configuring code page support	211
Defining the Web service client and wrapper programs.	211
Installing Web service support	211
Configuring the Web client	216
Running the Web service enabled application	218
Deploying the example application	222
Extracting the program interface	222
Running the Web services assistant program DFHLS2WS	224
An example of the generated WSDL document	225
Deploying the Web services binding file	227

Components of the base application	227
The catalog manager program.	230
File Structures and Definitions	234
Catalog file	234
Configuration file	235
Bibliography	237
The CICS Transaction Server for z/OS library	237
The entitlement set	237
PDF-only books	237
Other CICS books	239
Determining if a publication is current	239
Accessibility	241
Index	243
Notices	247
Trademarks.	248
Sending your comments to IBM	249

Preface

What this book is about

This book describes how to use Web Services in CICS®.

Who should read this book

This book is for:

- Planners and architects considering deploying CICS applications in a Web services environment.
- Systems programmers who are responsible for configuring CICS to support Web services
- Applications programmers who are responsible for applications that will be deployed in a Web services environment.

Chapter 1. CICS and Web Services

What the world wide web did for interactions between programs and end users, Web services can do for program-to-program interactions. Web services make it possible for applications to be integrated more rapidly, easily, and cheaply than ever before.

CICS Transaction Server for z/OS® provides comprehensive support for Web services:

- A CICS application can participate in a heterogeneous Web services environment as a service requester, as a service provider, or both.
- Support for HTTP and MQ
- CICS Transaction Server for z/OS includes the CICS Web services assistant, a set of utility programs that help you map WSDL service descriptions into high level programming language data structures, and *vice versa*. The utility programs support these programming languages:

COBOL

PL/I

C

C++

- The CICS support for Web services conforms to open standards including:
 - SOAP 1.1 and 1.2
 - HTTP 1.1
 - WSDL 1.1
- CICS support for Web services ensures maximum interoperability with other Web services implementations by conforming with the Web Services Interoperability Organization (WS-I) Basic Profile 1.1 Web Services Interoperability Organization (WS-I) Basic Profile 1.1 (<http://www.ws-i.org/Profiles/BasicProfile-1.1.html>) and the WS-I Simple SOAP Binding Profile 1.0 WS-I Simple SOAP Binding Profile 1.0 (<http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html>). The profiles are a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which, taken together, promote interoperability between different implementations of Web services. Conformance with both profiles is equivalent to conforming with the WS-I Basic Profile 1.0.

What is a Web service?

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically, Web Service Definition Language, or WSDL).

Web services fulfill a specific task or a set of tasks. A Web service is described using a standard, formal XML notion, called its service description, that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

The nature of the interface hides the implementation details of the service so that it can be used independently of the hardware or software platform on which it is implemented and independently of the programming language in which it is written.

This allows and encourages Web service based applications to be loosely coupled, component oriented, cross-technology implementations. Web services can be used alone or in conjunction with other Web services to carry out a complex aggregation or a business transaction.

How Web services can help your business

Web services is a technology for deploying, and providing access to, business functions over the World Wide Web. Web services make it possible for applications to be integrated more rapidly, easily, and cheaply than ever before.

Web services can help your business by:

- Reducing the cost of doing business
- Making it possible to deploy solutions more rapidly
- Opening up new opportunities.

The key to achieving all these things is a common program-to-program communication model, built on existing and emerging standards such as HTTP, XML, SOAP, and WSDL.

The support that CICS provides for Web services makes it possible for your existing applications to be deployed in new ways, with the minimum amount of reprogramming.

Web services terminology

Extensible Markup Language (XML)

A standard for document markup, which uses a generic syntax to mark up data with simple, human-readable tags. The standard is endorsed by the World Wide Web Consortium (W3C)World Wide Web Consortium (W3C) (<http://www.w3.org>).

Initial SOAP sender

The SOAP sender that originates a SOAP message at the starting point of a SOAP message path.

Service provider

The collection of software that provides a Web service.

Service provider application

An application that is used in a service provider. Typically, a service provider application provides the business logic component of a service provider.

Service requester

The collection of software that is responsible for requesting a Web service from a service provider.

Service requester application

An application that is used in a service requester. Typically, a service requester application provides the business logic component of a service requester.

Simple Object Access Protocol

See SOAP.

SOAP Formerly an acronym for *Simple Object Access Protocol*. A lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts:

- An envelope that defines a framework for describing what is in a message and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

SOAP can be used with other protocols, such as HTTP.

The specification for SOAP 1.1 is published at Simple Object Access Protocol (SOAP) 1.1 <http://www.w3.org/TR/SOAP>.

The specification for SOAP 1.2 is published at:

SOAP Version 1.2 Part 0: Primer <http://www.w3.org/TR/soap12-part0>

SOAP Version 1.2 Part 1: Messaging Framework <http://www.w3.org/TR/soap12-part1>

SOAP Version 1.2 Part 2: Adjuncts <http://www.w3.org/TR/soap12-part2>

SOAP intermediary

A SOAP node that is both a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.

SOAP message path

The set of SOAP nodes through which a single SOAP message passes. This includes the initial SOAP sender, zero or more SOAP intermediaries, and an ultimate SOAP receiver.

SOAP node

Processing logic which operates on a SOAP message.

SOAP receiver

A SOAP node that accepts a SOAP message.

SOAP sender

A SOAP node that transmits a SOAP message.

Ultimate SOAP receiver

The SOAP receiver that is a final destination of a SOAP message. It is responsible for processing the contents of the SOAP body and any SOAP header blocks targeted at it.

UDDI Universal Description, Discovery and Integration

Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI) is a specification for distributed Web-based information registries of Web services. UDDI is also a publicly accessible set of implementations of the specification that allow businesses to register information about the Web services they offer so that other businesses can find them. The specification is published by OASISOASIS (<http://www.oasis-open.org>)

Web service

A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically, Web Service Description Language, or WSDL).

Web Services Atomic Transaction

A specification that provides the definition of an atomic transaction coordination type used to coordinate activities having an "all or nothing" property.

The specification is published at <http://www.ibm.com/developerworks/library/specification/ws-tx/#atom><http://www.ibm.com/developerworks/library/specification/ws-tx/#atom>.

Web service binding file

A file, associated with a WEBSERVICE resource, which contains information that CICS uses to map data between input and output messages, and application data structures.

Web service description

An XML document by which a service provider communicates the specifications for invoking a Web service to a service requester. Web service descriptions are written in Web Service Description Language (WSDL).

Web Service Description Language

An XML application for describing Web services. It is designed to separate the descriptions of the abstract functions offered by a service, and the concrete details of a service, such as how and where that functionality is offered.

The specification is published at <http://www.w3.org/TR/wsdl><http://www.w3.org/TR/wsdl>.

Web Services Security

A set of enhancements to SOAP messaging that provides message integrity and confidentiality. The specification is published by OASIS (http://www.oasis-open.org) at Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>.

WS-Atomic Transaction

Web Services Atomic Transaction

WS-I Basic Profile

A set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which, taken together, promote interoperability between different implementations of Web services. The profile is defined by the Web Services Interoperability Organization (WS-I) and version 1.0 is available at Web Services Interoperability Organization (WS-I) Basic Profile 1.0 <http://www.ws-i.org/Profiles/BasicProfile-1.0.html>.

WSDL Web Service Description Language.

WSS Web Services Security

XML Extensible Markup Language.

The specifications for XML are published at:

SOAP Version 1.2 Part 0: Primer <http://www.w3.org/TR/soap12-part0>

SOAP Version 1.2 Part 1: Messaging Framework <http://www.w3.org/TR/soap12-part1>

SOAP Version 1.2 Part 2: Adjuncts <http://www.w3.org/TR/soap12-part2>

XML namespace

A collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

XML schema

An XML document that describes the structure, and constrains the contents of other XML documents.

XML schema definition language

An XML syntax for writing XML schemas, recommended by the World Wide Web Consortium (W3C).

Chapter 2. The Web services architecture

The Web services architecture is based upon interactions between three components: a service provider, a service requester, and an optional service registry.

The service provider

The collection of software that provides a Web service. It includes

- The application program
- The middleware
- The platform on which they run

The service requester

The collection of software that is responsible for requesting a Web service from a service provider. It includes

- The application program
- The middleware
- The platform on which they run

The service registry

A place where service providers publish descriptions of the services they provide, and where service requesters find them.

The registry is an optional component of the Web services architecture, as there are many situations where service requesters and providers can communicate without it. For example, the organization that provides a service can distribute the service description directly to the users of the service, using an attachment in an e-mail, or a download from an FTP site, or even a CD-ROM distribution.

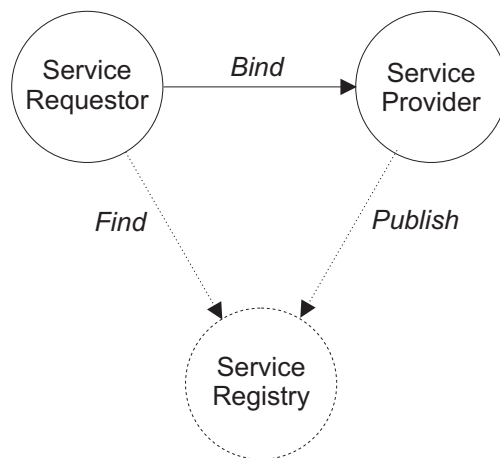


Figure 1. Web service components and interactions

CICS provides direct support for implementing the requester and provider components; you will need additional software to deploy a service registry in CICS. But because the Web service architecture is platform independent, you can, if you need a service registry, deploy it on another platform.

The interactions between the components involve the following operations:

Bind The service requester uses the service description to bind with the service provider and interact with the Web service implementation.

Publish

When a service registry is used, a service provider publishes its description in a registry so that the requester can find it.

Find When a service registry is used, a service requester finds the service description in the registry.

The Web service description

A Web service description is a document by which the *service provider* communicates the specifications for invoking the Web service to the *service requester*. Web service descriptions are expressed in the XML application known as Web Service Description Language (WSDL).

The service description describes the Web service in such a way as to minimize the amount of shared knowledge and customized programming that is needed to ensure communication between the service provider and the service requester. For example, neither the requester nor the provider needs to be aware of the platform on which the other runs, nor of the programming language in which the other is written.

The structure of WSDL allows a service description to be partitioned into:

- An abstract *service interface definition* that describes the interfaces of the service, and makes it possible to write programs that implement, and invoke, the service.
- A concrete *service implementation definition* that describes the location on the network (or *endpoint*) of the provider's Web service, and other implementation specific details, and that makes it possible for a service requester to connect to the service provider.

This is illustrated in Figure 2 on page 9.

A WSDL document uses the following major elements in the definition of network services:

<types>

A container for data type definitions using some type system (such as XML Schema). Defines the data types used within the message. The <types> element is not required when all messages consist of simple data types.

<message>

Specifies which XML data types are used to define the input and output parameters of an operation.

<portType>

Defines the set of operations supported by one or more endpoints. Within a <portType> element, each operation is described by an <operation> element.

<operation>

Specifies which XML messages can appear in the input and output data flows. An operation is comparable with a method signature in a programming language.

<binding>

Describes the protocol, data format, security and other attributes for a particular <portType> element.

<port> Specifies the network address of an endpoint, and associates it with a <binding> element.

<service>

Defines the Web service as a collection of related endpoints. A <service> element contains one or more <port> elements.

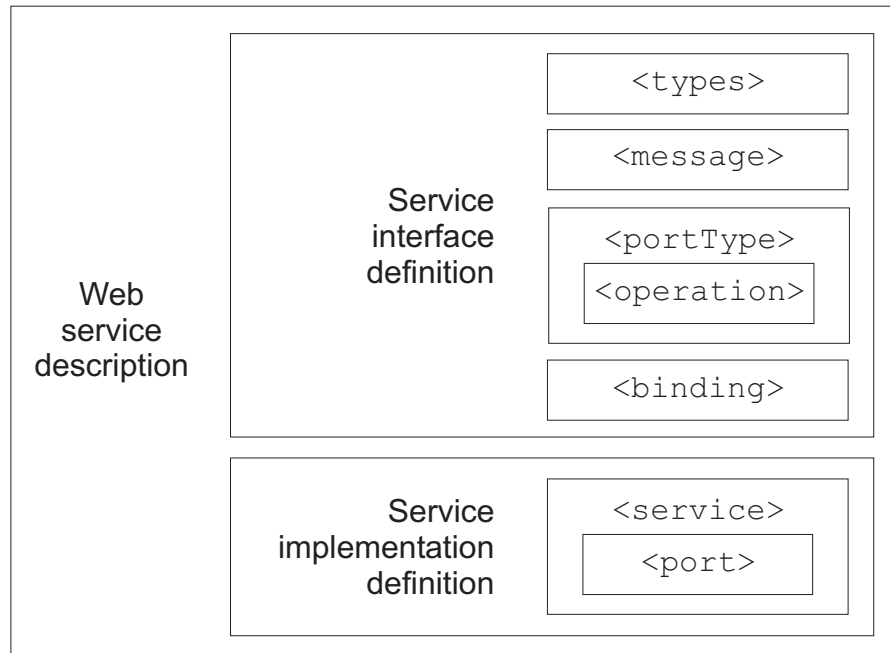


Figure 2. Structure of a Web service description

The ability to partition the Web service description makes it possible to divide responsibility for creating a complete service description. As an illustration, consider a service which is defined by a standards body for use across an industry, and implemented by individual companies within that industry:

- The standards body provides a service interface definition, containing the following elements:

- <types>
 - <message>
 - <portType>
 - <binding>

- A service provider who wishes to offer an implementation of the service provides a service implementation definition, containing the following elements:

- <port>
 - <service>

Service publication

A service description can be published using a number of different mechanisms; each mechanism has different capabilities, and is suitable for use in different situations. When necessary, a service description can be published in more than one way. Although CICS does not provide direct support for service publication, any of the mechanisms described can be used with CICS.

Direct publishing

This is the simplest mechanism for publishing service descriptions: the service provider sends the service description directly to the service requester. Ways to accomplish this include using an e-mail attachment, an FTP site, or a CD ROM distribution.

Advertisement and Discovery of Services (ADS)

DISCO

These proprietary protocols provide a dynamic publication mechanism. The service requester uses a simple HTTP GET mechanism to retrieve a Web service descriptions from a network location that is specified by the service provider, and identified with a URL.

Universal Description, Discovery and Integration (UDDI)

A specification for distributed Web-based information registries of Web services. UDDI is also a publicly accessible set of implementations of the specification that allow businesses to register information about the Web services they offer so that other businesses can find them.

Chapter 3. What is SOAP?

SOAP is a protocol for the exchange of information in a distributed environment. SOAP messages are encoded as XML documents, and can be exchanged using a variety of underlying protocols.

Formerly an acronym for *Simple Object Access Protocol*, SOAP is developed by the World Wide Web Consortium (W3C), and is defined in the following documents issued by W3C. You should consult these documents if you need complete, and authoritative, information about SOAP.

Simple Object Access Protocol (SOAP) 1.1 (W3C note)

SOAP Version 1.2 Part 0: Primer (W3C recommendation)

SOAP Version 1.2 Part 1: Messaging Framework (W3C recommendation)

SOAP Version 1.2 Part 2: Adjuncts (W3C recommendation)

The SOAP specifications describe a distributed processing model in which a *SOAP message* is passed between *SOAP nodes*. The message originates at a *SOAP sender* and is sent to a *SOAP receiver*. Between the sender and the receiver, the message may be processed by one or more *SOAP intermediaries*.

A SOAP message is a one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver, but messages can be combined to construct more complex interactions, such as request and response, and peer-to-peer conversations.

The specification also describes

- a set of encoding rules for expressing instances of application-defined data types.
- a convention for representing remote procedure calls and responses.

The structure of a SOAP message

A SOAP message is encoded as an XML document, consisting of an `<Envelope>` element, which contains an optional `<Header>` element, and a mandatory `<Body>` element. The `<Fault>` element, contained within the `<Body>`, is used for reporting errors.

The SOAP envelope

The SOAP `<Envelope>` is the root element in every SOAP message, and contains two child elements, an optional `<Header>` and a mandatory `<Body>`.

The SOAP header

The SOAP `<Header>` is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path.

The SOAP body

The SOAP `<Body>` is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

The SOAP fault

The SOAP `<Fault>` is a sub-element of the SOAP body, which is used for reporting errors.

With the exception of the <Fault> element, which is contained in the <Body> of a SOAP message, XML elements within the <Header> and the <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure.

Figure 3 shows the main elements of a SOAP message.

Figure 4 on page 13 is an example of a SOAP message that contains header

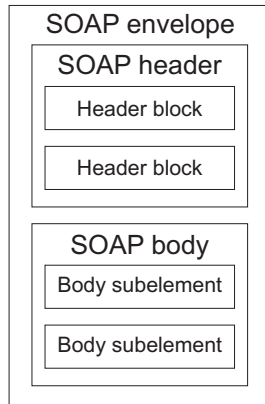


Figure 3. The structure of a SOAP message

blocks (the <m:reservation> and <n:passenger> elements) and a body (containing the <p:itinerary> and <q:lodging> elements).


```

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Åke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary
      xmlns:p="http://travelcompany.example.org/reservation/travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2001-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2001-12-20</p:departureDate>
        <p:departureTime>mid-morning</p:departureTime>
        <p:seatPreference/>
      </p:return>
    </p:itinerary>
    <q:lodging
      xmlns:q="http://travelcompany.example.org/reservation/hotels">
      <q:preference>none</q:preference>
    </q:lodging>
  </env:Body>
</env:Envelope>

```

Figure 4. An example of a SOAP 1.2 message

The SOAP header

The SOAP <Header> is an optional element within a SOAP message. It is used to pass application-related information that is to be processed by SOAP nodes along the message path.

The immediate child elements of the <Header> element are called header blocks; a header block is an application-defined XML element, and represents a logical grouping of data which can be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver.

SOAP header blocks can be processed by SOAP intermediary nodes, and by the ultimate SOAP receiver node; however, in a real application, not every node will process every header block. Rather, each node is typically designed to process particular header blocks, and - conversely - each header block is intended to be processed by particular nodes.

The SOAP header allows features to be added to a SOAP message in a decentralized manner without prior agreement between the communicating parties.

SOAP defines a few attributes that can be used to indicate who should deal with a feature and whether it is optional or mandatory. Such "control" information includes, for example, passing directives or contextual information related to the processing of the message. This allows a SOAP message to be extended in an application-specific manner.

Although the header blocks are application-defined, SOAP-defined attributes on the header blocks indicate how the header blocks are to be processed by the SOAP nodes. Some of the important attributes are:

encodingStyle

Indicates the rules used to encode the parts of a SOAP message: SOAP defines a narrower set of rules for encoding data than the very flexible encoding that XML allows.

role (SOAP 1.2)

actor (SOAP 1.1)

In SOAP 1.2, the **role** attribute specifies whether a particular node will operate on a message. If the role specified for the node matches the role attribute of the header block, the node processes the header; if the roles do not match, the node does not process the header block. In SOAP 1.1, the **actor** attribute performs the same function.

Roles can be defined by the application, and are designated by a URI. For example, `http://example.com/Log` might designate the role of a node which performs logging. Header blocks which are to be processed by this node specify `env:role="http://example.com/Log"` (where the namespace prefix `env` is associated with the SOAP namespace name of `http://www.w3.org/2003/05/soap-envelope`).

The SOAP 1.2 specification defines three standard roles in addition to those which are defined by the application:

`http://www.w3.org/2003/05/soap-envelope/none`

None of the SOAP nodes on the message path should process the header block directly. Header blocks with this role can be used to carry data that is required for processing of other SOAP header blocks.

`http://www.w3.org/2003/05/soap-envelope/next`

All SOAP nodes on the message path are expected to examine the header block (provided that the header has not been removed by a node earlier in the message path).

`http://www.w3.org/2003/05/soap-envelope/ultimateReceiver`

Only the ultimate receiver node is expected to examine the header block.

mustUnderstand

This attribute is used to ensure that SOAP nodes do not ignore header blocks which are important to the overall purpose of the application. If a SOAP node determines (using the **role** or **actor** attribute) that it should process a header block, and the **mustUnderstand** attribute has a value of "true", then the node must either process the header block in a manner consistent with its specification, or not at all (and throw a fault). But if the attribute has a value of "false", the node is not obliged to process the header block.

In effect, the **mustUnderstand** attribute indicates whether processing of the header block is mandatory or optional.

Values of the **mustUnderstand** attribute are:

true (SOAP 1.2)

1 (SOAP 1.1)

the node must either process the header block in a manner consistent with its specification, or not at all (and throw a fault).

false (SOAP 1.2)

0 (SOAP 1.1)

the node is not obliged to process the header block.

relay (SOAP 1.2 only)

When a SOAP intermediary node processes a header block, it removes it from the SOAP message. By default, it also removes any header blocks that it ignored (because the **mustUnderstand** attribute had a value of "false"). However, when the **relay** attribute is specified with a value of "true", the node retains the unprocessed header block in the message.

The SOAP body

The <Body> is the mandatory element within the SOAP envelope in which the main end-to-end information conveyed in a SOAP message is carried.

The <Body> element and its associated child elements are used to exchange information between the initial SOAP sender and the ultimate SOAP receiver. SOAP defines one child element for the <Body>: the <Fault> element is used for reporting errors. Other elements within the <Body> are defined by the Web service that uses them.

The SOAP fault

The SOAP <Fault> element is used to carry error and status information within a SOAP message.

If present, the SOAP <Fault> element must appear as a body entry and must not appear more than once within a Body element. The subelements of the SOAP <Fault> element are different in SOAP 1.1 and SOAP 1.2.

SOAP 1.1

In SOAP 1.1, the SOAP <Fault> element contains the following subelements:

<faultcode>

The <faultcode> element is a mandatory element within the <Fault> element. It provides information about the fault in a form that can be processed by software. SOAP defines a small set of SOAP fault codes covering basic SOAP faults, and this set can be extended by applications.

<faultstring>

The <faultstring> element is a mandatory element within the <Fault> element. It provides information about the fault in a form intended for a human reader.

<faultactor>

The <faultactor> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <faultactor> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but may do so.

<detail>

The <detail> element carries application-specific error information related to the <Body> element. It must be present if the contents of the <Body> element could not be successfully processed; it must not be used to carry

information about error information belonging to header entries - detailed error information belonging to header entries must be carried within header entries.

SOAP 1.2

In SOAP 1.2, the SOAP <Fault> element contains the following subelements:

<Code> The <Code> element is a mandatory element within the <Fault> element. It provides information about the fault in a form that can be processed by software. It contains a <Value> element and an optional <Subcode> element.

<Reason>

The <Reason> element is a mandatory element within the <Fault> element. It provides information about the fault in a form intended for a human reader. The <Reason> element contains one or more <Text> elements, each of which contains information about the fault in a different language.

<Node> The <Node> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <Node> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but may do so.

<Role> The <Role> element contains a URI that identifies the role the node was operating in at the point the fault occurred.

<Detail>

The <Detail> element is an optional element, which contains application-specific error information related to the SOAP fault codes describing the fault. The presence of the <Detail> element has no significance as to which parts of the faulty SOAP message were processed.

SOAP nodes

A SOAP node is the processing logic which operates on a SOAP message.

A SOAP node can:

- transmit a SOAP message
- receive a SOAP message
- process a SOAP message
- relay a SOAP message.

A SOAP node can be:

SOAP sender

A SOAP node that transmits a SOAP message.

SOAP receiver

A SOAP node that accepts a SOAP message.

Initial SOAP sender

The SOAP sender that originates a SOAP message at the starting point of a SOAP message path.

SOAP intermediary

A SOAP intermediary is both a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.

Ultimate SOAP receiver

The SOAP receiver that is a final destination of a SOAP message. It is responsible for processing the contents of the SOAP body and any SOAP header blocks targeted at it. In some circumstances, a SOAP message might not reach an ultimate SOAP receiver, for example because of a problem at a SOAP intermediary.

The SOAP message path

The SOAP message path is the set of SOAP nodes through which a single SOAP message passes. This includes the initial SOAP sender, zero or more SOAP intermediaries, and an ultimate SOAP receiver

In the simplest case, a SOAP message is transmitted between two nodes, that is from a *SOAP sender* to a *SOAP receiver*. However, in more complex cases, messages can be processed by *SOAP intermediary* nodes, which receive a SOAP message, and then send it to the next node. Figure 5 shows an example of a SOAP message path, in which a SOAP message is transmitted from the initial SOAP sender node, to the ultimate SOAP receiver node, passing through two SOAP intermediary nodes on its route.

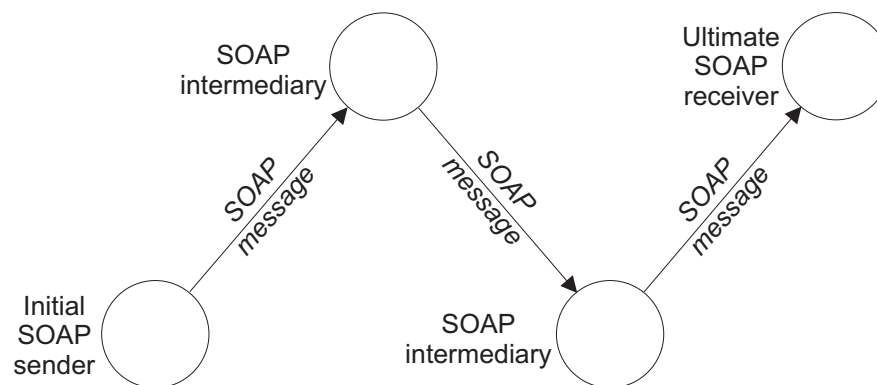


Figure 5. An example of a SOAP message path

A SOAP intermediary is both a SOAP receiver and a SOAP sender. It can (and in some cases must) process the header blocks in the SOAP message, and it forwards the SOAP message towards its ultimate receiver.

The *ultimate SOAP receiver* is the final destination of a SOAP message. As well as processing the header blocks, it is responsible for processing the SOAP body. In some circumstances, a SOAP message might not reach an ultimate SOAP receiver, for example because of a problem at a SOAP intermediary.

Chapter 4. How CICS supports Web services

CICS supports two different approaches to deploying your CICS applications in a Web services environment. One approach enables rapid deployment, with the least amount of programming effort; the other approach gives you complete flexibility and control over your Web service applications, using code that you write to suit your particular needs. Both approaches are underpinned by an infrastructure consisting of one or more *pipelines* and *message handler* programs which operate on Web service requests and responses.

When you deploy your CICS applications in a Web services environment:

- You can use the CICS Web services assistant to help you deploy an application with the least amount of programming effort.

For example, if you want to expose an existing application as a Web service, you can start with a high-level language data structure, and generate the Web services description. Alternatively, if you want to communicate with an existing Web service, you can start with its Web service description and generate a high level language structure that you can use in your program.

The CICS Web services assistant also generates the CICS resources that you need to deploy your application. And when your application runs, CICS transforms your application data into a SOAP message on output, and transforms the SOAP message back to application data on input.

- You can take complete control over the processing of your data by writing your own code to map between your application data and the message that flows between the service requester and provider.

For example, if you want to use non-SOAP messages within the Web service infrastructure, you can write your own code to transform between the message format and the format used by your application.

Whichever approach you follow, you can use your own message handlers to perform additional processing on your request and response messages, or use CICS-supplied message handlers which are designed especially to help you process SOAP messages.

Message handlers and pipelines

A *message handler* is a program in which you can perform your own processing of Web service requests and responses. A *pipeline* is a set of message handlers that are executed in sequence.

There are two distinct phases in the operation of a pipeline:

1. The *request phase*, during which CICS invokes each handler in the pipeline in turn. Each message handler can process the request before returning control to CICS.
2. This is followed by the *response phase*, during which CICS again invokes each handler in turn, but with the sequence reversed. That is, the message handler that is invoked first in the request phase, is invoked last in the response phase. Each message handler can process the response during this phase.

Not every request is succeeded by a response; some applications use a one-way message flow from service requester to provider. In this case, although there is no message to be processed, each handler is invoked in turn during the response phase.

Figure 6 shows a pipeline of three message handlers:

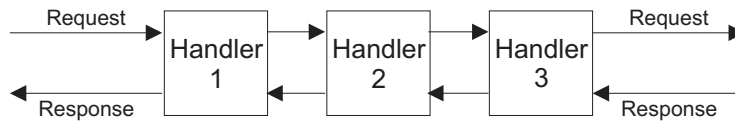


Figure 6. A generic CICS pipeline

In this example, the handlers are executed in the following sequence:

In the request phase

1. Handler 1
2. Handler 2
3. Handler 3

In the response phase

1. Handler 3
2. Handler 2
3. Handler 1

In a service provider, the transition between the phases normally occurs in the last handler in the pipeline (known as the *terminal handler*) which absorbs the request, and generates a response; in a service requester, the transition occurs when the request is processed in the service provider. However, a message handler in the request phase can force an immediate transition to the response phase, and an immediate transition can also occur if CICS detects an error.

A message handler can modify the message, or can leave it unchanged. For example:

- A message handler that performs encryption and decryption will receive an encrypted message on input, and pass the decrypted message to the next handler. On output, it will do the opposite: receive a plain text message, and pass an encrypted version to the following handler.
- A message handler that performs logging will examine a message, and copy the relevant information from that message to the log. The message that is passed to the next handler is unchanged.

Important: If you are familiar with the SOAP feature for CICS TS, you should be aware that the structure of the pipeline in this release of CICS is not the same as that used in the feature.

Transport-related handlers

CICS supports the use of two transport mechanisms between the Web service requester and the provider. In some cases, you might require different message handlers to be invoked, depending upon which transport mechanism is in use. For example, you might wish to include message handlers that perform encryption of parts of your messages when you are using the HTTP transport to communicate on an external network. But encryption might not be required when you are using the MQ transport on a secure internal network.

To support this, you can configure your pipeline to specify handlers that are invoked only when a particular transport (HTTP or MQ) is in use. For a service provider, you can be even more specific, and specify handlers that are invoked only when a particular named resource (a TCPIP SERVICE for the HTTP transport, a QUEUE for the MQ transport) is in use.

This is illustrated in Figure 7:

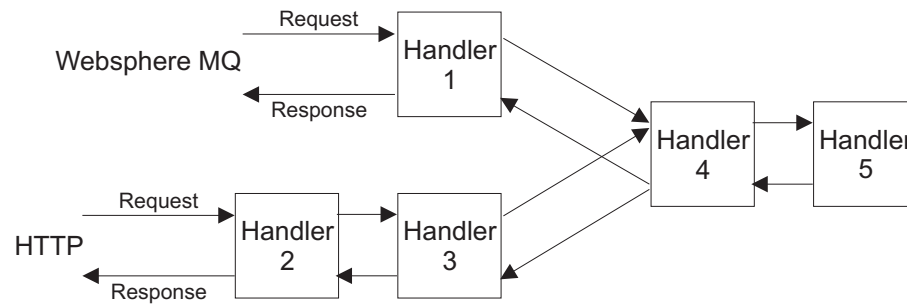


Figure 7. Pipeline with transport-related handlers

In this example, which applies to a service provider:

- Handler 1 is invoked for messages that use the MQ transport.
- Handlers 2 and 3 are invoked for messages that use the HTTP transport.
- Handlers 4 and 5 are invoked for all messages.
- Handler 5 is the terminal handler.

Interrupting the flow

During processing of a request, a message handler can decide not to pass a message to the next handler, but can, instead, generate a response. Normal processing of the message is interrupted, and some handlers in the pipeline are not invoked. For example, suppose that handler 2 in Figure 8 is responsible for performing security checks.

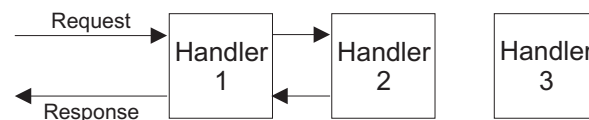


Figure 8. Interrupting the pipeline flow

If the request does not bear the correct security credentials, then, instead of passing the request to handler 3, handler 2 suppresses the request and constructs a suitable response. The pipeline is now in the response phase, and when handler 2 returns control to CICS, the next handler invoked is handler 1, and handler 3 is bypassed altogether.

A handler that interrupts the normal message flow in this way must only do so if the originator of the message expects a response; for example, a handler should not generate a response when an application uses a one-way message flow from service requester to provider.

A service provider pipeline

In a service provider pipeline, CICS receives a request, which is passed through a pipeline to the target application program. The response from the application is returned to the service requester through the same pipeline.

When CICS is in the role of service provider, it performs the following operations:

1. Receive the request from the service requester.

2. Examine the request, and extract the contents that are relevant to the target application program.
3. Invoke the application program, passing data extracted from the request.
4. When the application program returns control, construct a response, using data returned by the application program.
5. Send a response to the service requester.

Figure 9 illustrates a pipeline of three message handlers in a service provider setting:

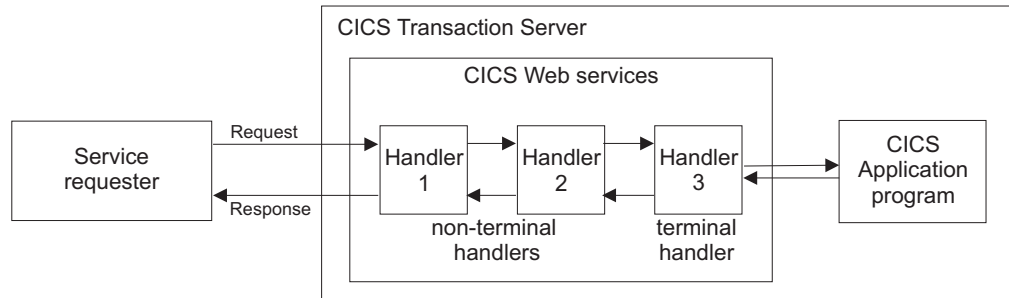


Figure 9. A service provider pipeline

1. CICS receives a request from the service requester. It passes the request to message handler 1.
2. Message handler 1 performs some processing, and passes the request to handler 2 (To be precise, it returns control to CICS, which manages the pipeline. CICS then passes control to the next message handler).
3. Message handler 2 receives the request from handler 1, performs some processing, and passes the request to handler 3.
4. Message handler 3 is the terminal handler of the pipeline. It uses the information in the request to invoke the application program. It then uses the output from the application program to generate a response, which it passes back to handler 2.
5. Message handler 2 receives the response from handler 3, performs some processing, and passes it to handler 1.
6. Message handler 1 receives the response from handler 2, performs some processing, and returns the response to the service requester.

A service requester pipeline

In a service requester pipeline, an application program creates a request, which is passed through a pipeline to the service provider. The response from the service provider is returned to the application program through the same pipeline.

When CICS is in the role of service requester, it performs the following operations:

1. Use data provided by the application program to construct a request.
2. Send the request to the service provider.
3. Receive a response from the service provider.
4. Examine the response, and extract the contents that are relevant to the original application program.
5. Return control to the application program.

Figure 10 illustrates a pipeline of three message handlers in a service requester setting:

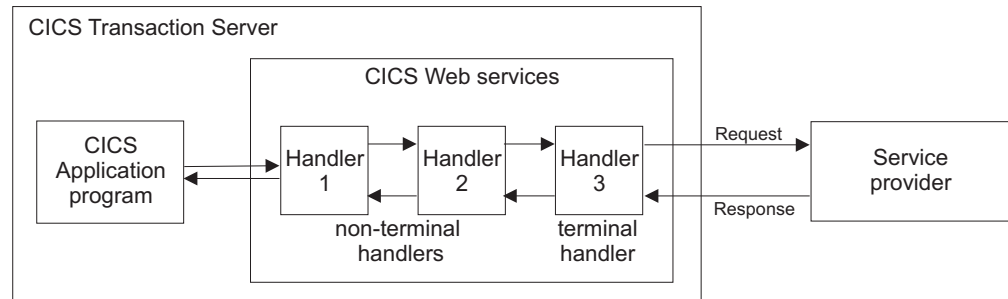


Figure 10. A service requester pipeline

1. An application program creates a request.
2. Message handler 1 receives the request from the application program, performs some processing, and passes the request to handler 2 (To be precise, it returns control to CICS, which manages the pipeline. CICS then passes control to the next message handler).
3. Message handler 2 receives the request from handler 1, performs some processing, and passes the request to handler 3.
4. Message handler 3 receives the request from handler 2, performs some processing, and passes the request to the service provider.
5. Message handler 3 receives the response from the service provider, performs some processing, and passes it to handler 2.
6. Message handler 2 receives the response from handler 3, performs some processing, and passes it to handler 1.
7. Message handler 1 receives the response from handler 2, performs some processing, and returns the response to the application program.

CICS pipelines and SOAP

The pipeline which CICS uses to process Web service requests and responses is generic, in that there are few restrictions on what processing can be performed in each message handler. However, many Web service applications use SOAP messages, and any processing of those messages should comply with the SOAP specification. Therefore, CICS provides special *SOAP message handler* programs that can help you to configure your pipeline as a SOAP node.

- Your pipeline can be configured to support SOAP 1.1 or SOAP 1.2. Within your CICS system, you can have many pipelines, some of which support SOAP 1.1 and some of which support SOAP 1.2.
- A pipeline can be configured for use in a service requester, or in a service provider:
 - A service requester pipeline is the initial SOAP sender for the request, and the ultimate SOAP receiver for the response
 - A service provider pipeline is the ultimate SOAP receiver for the request, and the initial SOAP sender for the response

You cannot configure a CICS pipeline to function as a SOAP intermediary.

- You can configure a CICS pipeline to have more than one SOAP message handler.

- The CICS-provided SOAP message handlers can be configured to invoke one or more user-written header-handling routines.
- The CICS-provided SOAP message handlers can be configured to enforce some aspects of compliance with the WS-I Basic Profile 1.1 and Simple SOAP Binding Profile 1.0, and to enforce the presence of particular headers in the SOAP message.

The SOAP message handlers, and their header handling routines are specified in the pipeline configuration file.

SOAP messages and the application data structure

In many cases, the CICS Web services assistant can generate the code to transform the data between a high level data structure used in an application program, and the contents of the <Body> element of a SOAP message. In these cases, when you write your application program, you do not need to parse or construct the SOAP body; CICS will do this for you.

In order to transform the data, CICS needs information, at run time, about the application data structure, and about the format of the SOAP messages. This information is held in two files:

- The Web service binding file

This file is generated by the CICS Web services assistant from an application language data structure, using utility program DFHLS2WS, or from a Web service description, using utility program DFHWS2LS. CICS uses the binding file to generate the resources used by the Web service application, and to perform the mapping between the application's data structure and the SOAP messages.

- The Web service description

This may be an existing Web service description, or it may be generated from an application language data structure, using utility program DFHLS2WS. CICS uses the Web service description to perform full validation of SOAP messages.

Figure 11 shows where these files are used in a service provider.

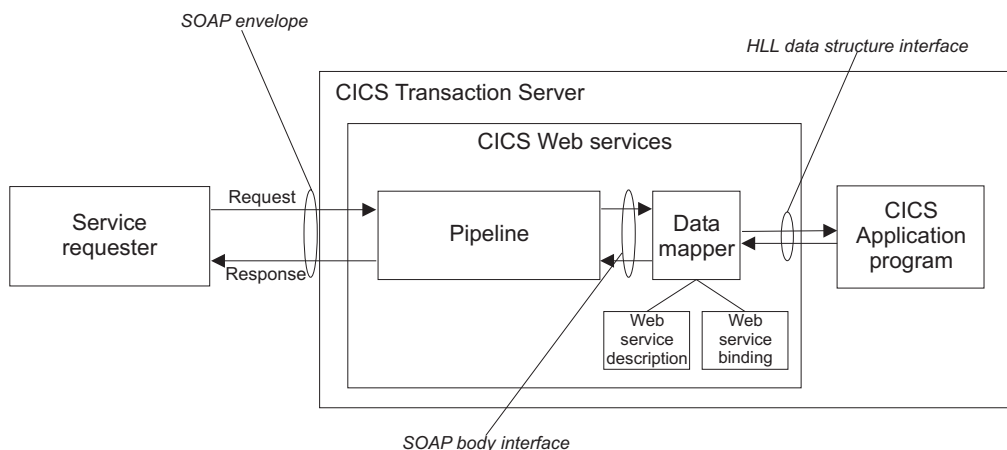


Figure 11. Mapping the SOAP body to the application data structure in a service provider

A message handler in the pipeline (typically, a CICS-supplied SOAP message handler) removes the SOAP envelope from an inbound request, and passes the SOAP body to the data mapper function. This uses the Web service binding file to map the contents of the SOAP body to the application's data structure. If full

validation of the SOAP message is active, then the SOAP body is validated against the Web service description. If there is an outbound response, the process is reversed.

Figure 12 shows where these files are used in a service requester.

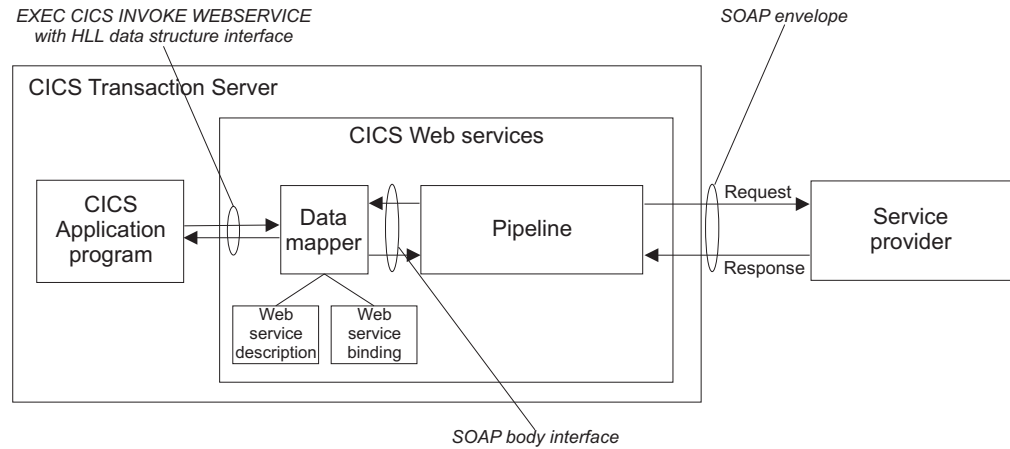


Figure 12. Mapping the SOAP body to the application data structure in a service requester

For an outbound request, the data mapper function constructs a SOAP body from the application's data structure, using information from the Web service binding file. A message handler in the pipeline (typically, a CICS-supplied SOAP message handler) adds the SOAP envelope. If there is an inbound response, the process is reversed. If full validation of the SOAP message is active, then the inbound SOAP body is validated against the Web service description.

In both cases, the execution environment that allows a particular CICS application program to operate in a Web services setting is defined by three objects. These are the pipeline, the Web service binding file, and the Web service description. The three objects are defined to CICS as attributes of the WEBSERVICE resource definition.

There are some situations in which, even though you are using SOAP messages, you cannot use the transformation that the CICS Web services assistant generates:

- When the same data cannot be represented in the SOAP message and in the high level language.

All the high level languages that CICS supports, and XML Schema, support a variety of different data types. However, there is not a one-to-one correspondence between the data types used in the high level languages, and those used in XML Schema, and there are cases where data can be represented in one, but not in the other. In this situations, you should consider one of the following:

 - Change your application data structure. This may not be feasible, as it might entail changes to the application program itself.
 - Construct a wrapper program, which transforms the application data into a form that CICS can then transform into a SOAP message body. If you do this, you can leave your application program unchanged. In this case CICS Web service support interacts directly with the wrapper program, and only indirectly with the application program.
- When your application program is in a language which is not supported by the CICS Web services assistant.

In this situation, you should consider one of the following:

- Construct a wrapper program that is written in one of the languages that the CICS Web services assistant does support (COBOL, PL/I, C or C++).
- Instead of using the CICS Web services assistant, write your own program to perform the mapping between the SOAP messages and the application program's data structure.

WSDL and the application data structure

A Web service description contains abstract representations of the input and output messages used by the service. CICS uses the Web service description to construct the data structures used by application programs. At run time, CICS performs the mapping between the application data structures and the messages.

The description of a Web service contains, among other things:

- One or more operations
- For each operation, an input message and an optional output message
- For each message, the message structure, defined in terms of XML data types. Complex data types used in the messages are defined in an XML schema which is contained in the <types> element within the Web service description. Simple messages can be described without using the <types> element.

WSDL contains an abstract definition of an operation, and the associated messages; it cannot be used directly in an application program. To implement the operation, a service provider must do the following:

- It must parse the WSDL, in order to understand the structure of the messages
- It must parse each input message, and construct the output message
- It must perform the mappings between the contents of the input and output messages, and the data structures used in the application program

A service requester must do the same in order to invoke the operation.

When you use the the CICS Web services assistant, much of this is done for you, and you can write your application program without detailed understanding of WSDL, or of the way the input and output messages are constructed.

The CICS Web services assistant consists of two utility programs:

DFHWS2LS

This utility program takes a Web service description as a starting point. It uses the descriptions of the messages, and the data types used in those messages, to construct high level language data structures that you can use in your application programs.

DFHLS2WS

This utility program takes a high level language data structure as a starting point. It uses the structure to construct a Web services description that contains descriptions of messages, and the data types used in those messages derived from the language structure.

Both utility programs generate a Web services binding file that CICS uses at run time to perform the mapping between the application program's data structures and the SOAP messages.

An example of COBOL to WSDL mapping

This example shows how the data structure used in a COBOL program is represented in the Web services description that is generated by the CICS Web services assistant.

Figure 13 shows a simple COBOL data structure:

```
*   Catalogue COMMAREA structure
      03 CA-REQUEST-ID           PIC X(6).
      03 CA-RETURN-CODE         PIC 9(2).
      03 CA-RESPONSE-MESSAGE    PIC X(79).
*   Fields used in Place Order
      03 CA-ORDER-REQUEST.
          05 CA-USERID           PIC X(8).
          05 CA-CHARGE-DEPT      PIC X(8).
          05 CA-ITEM-REF-NUMBER  PIC 9(4).
          05 CA-QUANTITY-REQ     PIC 9(3).
          05 FILLER              PIC X(888).
```

Figure 13. COBOL record definition of an input message defined in WSDL

The key elements in the corresponding fragment of the Web services description are shown in Figure 14 on page 28:

```

<xsd:sequence>
  <xsd:element name="CA-REQUEST-ID" nillable="false">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:length value="6"/>
        <xsd:whiteSpace value="preserve"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="CA-RETURN-CODE" nillable="false">
    <xsd:simpleType>
      <xsd:restriction base="xsd:short">
        <xsd:maxInclusive value="99"/>
        <xsd:minInclusive value="0"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="CA-RESPONSE-MESSAGE" nillable="false">
    ...
  </xsd:element>
  <xsd:element name="CA-ORDER-REQUEST" nillable="false">
    <xsd:complexType mixed="false">
      <xsd:sequence>
        <xsd:element name="CA-USERID" nillable="false">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:length value="8"/>
              <xsd:whiteSpace value="preserve"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="CA-CHARGE-DEPT" nillable="false">
          ...
        </xsd:element>
        <xsd:element name="CA-ITEM-REF-NUMBER" nillable="false">
          ...
        </xsd:element>
        <xsd:element name="CA-QUANTITY-REQ" nillable="false">
          ...
        </xsd:element>
        <xsd:element name="FILLER" nillable="false">
          ...
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:sequence>

```

Figure 14. WSDL fragment derived from a COBOL data structure

The Web service binding file

The *Web service binding file* contains information that CICS uses to map data between input and output messages, and application data structures.

A Web service description contains abstract representations of the input and output messages used by the service. When a service provider or service requester application executes, CICS needs information about how the contents of the messages maps to the data structures used by the application. This information is held in a Web service binding file.

Web service binding files are created:

- By utility program DFHWS2LS when language structures are generated from WSDL.
- By utility program DFHLS2WS when WSDL is generated from a language structure.

At run time, CICS uses information in the Web service binding file to perform the mapping between application data structures and SOAP messages. Web service binding files are defined to CICS in the WSBIND attribute of the WEBSERVICEWEBSERVICE resource.

External standards

CICS support for Web services conforms to a number of industry standards and specifications.

Extensible Markup Language Version 1.0

Extensible Markup Language (XML) 1.0 is a subset of SGML. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML.

XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

The specification for XML 1.0 and its errata is published by the World Wide Web Consortium (W3C) as a W3C Recommendation at <http://www.w3.org/TR/REC-xml> XML Version 1.0.

SOAP 1.1 and 1.2

SOAP is a lightweight, XML-based, protocol for exchange of information in a decentralized, distributed environment.

The protocol consists of three parts:

- An envelope that defines a framework for describing what is in a message and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

SOAP can be used with other protocols, such as HTTP.

The specifications for SOAP are published by the World Wide Web Consortium (W3C). The specification for SOAP 1.1 is described as a note at <http://www.w3.org/TR/SOAPSimple> Object Access Protocol 1.1. This specification has not been endorsed by the W3C, but forms the basis for the SOAP 1.2 specification. It expands the SOAP acronym to Simple Object Access Protocol.

SOAP 1.2 is a W3C recommendation and is published in two parts:

- Part 1: Messaging Framework Part 1: Messaging Framework is published at <http://www.w3.org/TR/soap12-part1/>.
- Part 2: Adjuncts Part 2: Adjuncts is published at <http://www.w3.org/TR/soap12-part2/>.

The specification also includes a primer that is intended to provide a tutorial on the features of the SOAP Version 1.2 specification, including usage scenarios. The primer is published at <http://www.w3.org/TR/soap12-part0/> SOAP 1.2 Primer. The specification for SOAP 1.2 does not expand the acronym.

Web Services Description Language Version 1.1

Web Services Description Language (WSDL) 1.1 is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete end points are combined into abstract endpoints (services).

WSDL is extensible to allow the description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. The WSDL 1.1 specification only defines bindings that describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET and POST, and MIME.

The specification for WSDL is published by the World Wide Web Consortium (W3C) as a W3C Note at <http://www.w3.org/TR/wsdlWSDL> Version 1.1.

Web Services Coordination Version 1.0

Web Services Coordination Version 1.0 (or WS-Coordination) is an extensible framework for providing protocols that coordinate the actions of distributed applications. These coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed activities.

The framework enables an application service to create a context needed to propagate an activity to other services and to register for coordination protocols. The framework enables existing transaction processing, workflow, and other systems for coordination to hide their proprietary protocols and to operate in a heterogeneous environment.

The specification for WS-Coordination is published at Web Services Transactions specifications <http://www.ibm.com/developerworks/library/specification/ws-tx/>.

Web Services Atomic Transaction Version 1.0

Web Services Atomic Transaction Version 1.0 (or WS-AtomicTransaction) is a protocol that defines the atomic transaction coordination type for transactions of a short duration. It is used with the extensible coordination framework described in the Web Services Coordination Version 1.0 (or WS-Coordination) specification.

The WS-AtomicTransaction specification and the WS-Coordination specification define protocols for short term transactions that enable transaction processing systems to interoperate in a Web services environment. Transactions that use WS-AtomicTransaction have the *ACID* properties of atomicity, consistency, isolation, and durability.

The specification for WS-AtomicTransaction is published at Web Services Transactions specifications <http://www.ibm.com/developerworks/library/specification/ws-tx/>.

WS-I Basic Profile Version 1.1

WS-I Basic Profile Version 1.1 (WS-I BP 1.1) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which together promote interoperability between different implementations of Web services.

The WS-I BP 1.1 is derived from Basic Profile Version 1.0 by incorporating its published errata and separating out the requirements that relate to the serialization of envelopes and their representation in messages. These requirements are now part of the Simple SOAP Binding Profile Version 1.0.

To summarize, the WS-I Basic Profile Version 1.0 has now been split into two separately published profiles. These are:

- WS-I Basic Profile Version 1.1
- WS-I Simple SOAP Binding Profile Version 1.0

Together, these two Profiles supersede the WS-I Basic Profile Version 1.0.

The reason for this separation is to enable the Basic Profile 1.1 to be composed with any profile that specifies envelope serialization, including the Simple SOAP Binding Profile 1.0.

The specification for WS-I BP 1.1 is published by the Web Services Interoperability Organization (WS-I), and can be found at <http://www.ws-i.org/Profiles/BasicProfile-1.1.html> WS-I Basic Profile Version 1.1.

WS-I Simple SOAP Binding Profile Version 1.0

WS-I Simple SOAP Binding Profile Version 1.0 (SSBP 1.0) is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

The SSBP 1.0 is derived from the WS-I Basic Profile 1.0 requirements that relate to the serialization of the envelope and its representation in the message.

WS-I Basic Profile 1.0 has now been split into two separately published profiles. These are:

- WS-I Basic Profile Version 1.1
- WS-I Simple SOAP Binding Profile Version 1.0

Together, these two Profiles supersede the WS-I Basic Profile Version 1.0.

The specification for SSBP 1.0 is published by the Web Services Interoperability Organization (WS-I), and can be found at <http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html> WS-I Simple SOAP Binding Profile Version 1.0.

Web Services Security: SOAP Message Security

Web Services Security (WSS): SOAP Message Security is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. WSS: SOAP Message Security is extensible, and can accommodate a variety of security models and encryption technologies.

WSS: SOAP Message Security provides three main mechanisms that can be used independently or together. They are:

- The ability to send security tokens as part of a message, and for associating the security tokens with message content
- The ability to protect the contents of a message from unauthorized and undetected modification (message integrity)
- The ability to protect the contents of a message from unauthorized disclosure (message confidentiality).

WSS: SOAP Message Security can be used in conjunction with other Web service extensions and application-specific protocols to satisfy a variety of security requirements.

The specification is published by the Organization for the Advancement of Structured Information Standards (OASIS) at Web Services Security: SOAP Message Security 1.0 (WS-Security 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

Web Services Security (WSS): UsernameToken Profile 1.0 describes how to use the UsernameToken in conjunction with the WSS: SOAP Message Security specification. More specifically, it covers how a Web service can use a UsernameToken as a means of providing a username and password authentication between a Web service provider and requester.

The WSS: UsernameToken Profile 1.0 specification is published by the Organization for the Advancement of Structured Information Standards (OASIS) at Web Services Security: UsernameTokenProfile 1.0 (WS-Security 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 1.0

Web Services Security (WSS): X.509 Certificate Token Profile 1.0 describes how to use X.509 certificates in conjunction with the WSS: SOAP Message Security specification. More specifically, it covers how a Web service can use X.509 certificates as a means of providing authentication between a Web service provider and requester.

The WSS: X.509 Certificate Token Profile 1.0 specification is published by the Organization for the Advancement of Structured Information Standards (OASIS) at Web Services Security: X.509 Certificate Token Profile 1.0 (WS-Security 2004)<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>.

XML Encryption Syntax and Processing

XML Encryption Syntax and Processing specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.

XML Encryption Syntax and Processing is a recommendation of the World Wide Web Consortium (W3C) and is published at XML-Signature Syntax and Processing<http://www.w3.org/TR/xmlenc-core>.

XML-Signature Syntax and Processing

XML-Signature Syntax and Processing specifies processing rules and syntax for XML digital signatures.

XML digital signatures provide integrity, message authentication, and signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.

The specification for XML-Signature is published by World Wide Web Consortium (W3C) at XML-Signature Syntax and Processing<http://www.w3.org/TR/xmlsig-core>.

CICS compliance with Web services standards

CICS is compliant with the supported Web services standards and specifications, in that it allows you to generate and deploy Web services that are compliant.

It should be noted that CICS does not enforce this compliancy. For example, in the case of support for the WS-I Basic Profile 1.1 specification, CICS allows you to apply additional qualities of service to your Web service that could break the interoperability outlined in this Profile.

How CICS complies with WS-I Basic Profile 1.1

CICS conditionally complies with WS-I Basic Profile 1.1 in that it adheres to all the *MUST* level requirements. However, CICS does not specifically implement support for UDDI registries, and therefore the points relating to this in the specification are ignored. Also the Web services assistant jobs and associated runtime environment are not fully compliant with this Profile, as there are limitations in the support of mapping certain schema elements.

See “High level language and XML schema mapping” on page 83 for a list of unsupported schema elements.

Conformance targets identify what artifacts (e.g. SOAP message, WSDL description) or parties (e.g. SOAP processor, end user) that the requirements apply to. The conformance targets supported by CICS are:

MESSAGE

Protocol elements that transport the ENVELOPE (e.g. SOAP over HTTP messages).

ENVELOPE

The serialization of the soap:Envelope element and its content.

DESCRIPTION

The description of types, messages, interfaces and their protocol and data format bindings, and network access points associated with Web services (e.g. WSDL descriptions).

INSTANCE

Software that implements a wsdl:port.

CONSUMER

Software that invokes an INSTANCE.

SENDER

Software that generates a message according to the protocol associated with it

RECEIVER

Software that consumes a message according to the protocol associated with it.

How CICS complies with Web Services Security specifications

CICS conditionally complies with Web Services Security: SOAP Message Security and related specifications by supporting the following aspects.

Compliance with Web Services Security: SOAP Message Security

Security header

The <wsse:Security> header provides a mechanism for attaching security-related information targeted at a specific recipient in the form of a

SOAP actor or role. This could be the ultimate recipient of the message or an intermediary. The following attributes are supported in CICS:

- S11:actor (for an intermediary)
- S11:mustUnderstand
- S12:role (for an intermediary)
- S12:mustUnderstand

Security tokens

The following security tokens are supported in the security header:

- User name and password
- Binary security token (X.509 certificate)

Token references

A security token conveys a set of claims. Sometimes these claims reside elsewhere and need to be accessed by the receiving application. The <wssc:SecurityTokenReference> element provides an extensible mechanism for referencing security tokens. The following mechanisms are supported:

- Direct reference
- Key identifier
- Key name
- Embedded reference

Signature algorithms

This specification builds on XML Signature and therefore has the same algorithms as those that are specified as required in the XML Signature specification. CICS supports:

Algorithm type	Algorithm	URI
Digest	SHA1	http://www.w3.org/2000/09/xmlsig#sha1
Signature	DSA with SHA1 (validation only)	http://www.w3.org/2000/09/xmlsig#dsa-sha1
Signature	RSA with SHA1	http://www.w3.org/2000/09/xmlsig#rsa-sha1
Canonicalization	Exclusive XML canonicalization (without comments)	http://www.w3.org/2001/10/xml-exc-c14n#

Signature signed parts

CICS allows the following SOAP elements to be signed:

- the SOAP message body
- the identity token (a type of security token), that is used as an asserted identity

Encryption algorithms

The following data encryption algorithms are supported:

Algorithm	URI
Triple Data Encryption Standard algorithm (Triple DES)	http://www.w3.org/2001/04/xmlenc#tripledes-cbc

Algorithm	URI
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	http://www.w3.org/2001/04/xmlenc#aes128-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	http://www.w3.org/2001/04/xmlenc#aes192-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	http://www.w3.org/2001/04/xmlenc#aes256-cbc

The following key encryption algorithm is supported:

Algorithm	URI
Key transport (public key cryptography) RSA Version 1.5:	http://www.w3.org/2001/04/xmlenc#rsa-1_5

Encryption message parts

CICS allow the following SOAP elements to be encrypted:

- the SOAP body

Timestamp

The <wsu:Timestamp> element provides a mechanism for expressing the creation and expiration times of the security semantics in a message. CICS tolerates the use of timestamps within the Web services security header on inbound SOAP messages.

Error handling

CICS generates SOAP fault messages using the standard list of response codes listed in the specification.

Compliance with Web Services Security: UsernameToken Profile 1.0

The following aspects of this specification are supported:

Password types

Text

Token references

Direct reference

Compliance with Web Services Security: X.509 Certificate Token Profile 1.0

The following aspects of this specification are supported:

Token types

- X.509 Version 3: Single certificate. See <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3>.
- X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL). See <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1>.
- X.509 Version 3: PKCS7 with or without CRLs. The IBM® Software Development Kit (SDK) supports both. The Sun Java™ Development Kit (JDK) supports PKCS7 without CRL only.

Token references

- Key identifier - subject key identifier
- Direct reference
- Custom reference - issuer name and serial number

Aspects that are not supported

The following items are not supported in CICS:

- Validation of Timestamps for freshness
- Nonces
- Web services security for SOAP attachments
- Security Assertion Markup Language (SAML) token profile, WS-SecurityKerberos token profile, and XrML token profile
- Web Services Interoperability (WS-I) Basic Security Profile
- XML enveloping digital signature
- XML enveloping digital encryption
- The following transport algorithms for digital signatures are not supported:
 - XSLT: <http://www.w3.org/TR/1999/REC-xslt-19991116>
 - SOAP Message Normalization. For more information, see <http://www.w3.org/TR/2003/NOTE-soap12-n11n-20031008/>
- The Diffie-Hellman key agreement algorithm for encryption is not supported. For more information, see <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html#sec-DHKeyValue>.
- The following canonicalization algorithms for encryption, which are optional in the XML encryption specification, are not supported:
 - Canonical XML with or without comments
 - Exclusive XML canonicalization with or comments
- In the Username Token Version 1.0 Profile specification, the digest password type is not supported.

Chapter 5. Getting started with Web services

There are several ways to get started with Web services in CICS. Which is most appropriate for you will depend upon how much you already know about the subject, and upon how well advanced your plans are for using Web services.

Here are some starting points for Web services in CICS:

- Install the example application. CICS provides an example of a catalog management application which can be enabled as a Web service provider. The example includes all the code and resource definitions that you need to get the application working in CICS with the minimum amount of work. It also includes code to interact with the service that runs on a number of common Web service clients.

Use the example application if you want a rapid “proof-of-concept” demonstration that you can deploy a Web service in CICS, or if you want a “hands-on” way to learn about Web services in CICS.

The example application is described in “The CICS catalog manager example application” on page 43

- Get straight to work planning to deploy an application as a service provider or a requester. You may already know enough about how you will use Web services in CICS to start planning your applications and the related infrastructure.
- Migrate from the SOAP feature for CICS If you have an existing application that uses the feature, you may be ready to start planning how you will redeploy the application.

Planning to use Web services

Before you can plan to use Web services in CICS, you need to consider these questions for each application:

Do you plan to deploy your CICS application in the role of a service provider or a service requester?

You may have a pair of applications that you want to connect using CICS support for Web services. In this case, one application will be the service provider; the other will be the service requester.

Do you plan to use your existing application programs, or write new ones?

If your existing applications are designed with a well defined interface to the business logic, you will probably be able to use them in a Web services setting, either as a service provider or a service requester. However, in most cases, you will need to write a wrapper program that connects your business logic to the Web services logic.

If you plan to write new applications, you should aim to keep your business logic separated from your Web services logic, and, once again, you will need to write a wrapper program to provide this separation. However, if your application is designed with Web services in mind, the wrapper may prove to be simpler to write.

Do you intend to use SOAP messages?

SOAP is fundamental to the Web services architecture, and much of the support that is provided in CICS assumes that you will use SOAP. However, there may be situations where you wish to use other message formats. For example, you may have developed your own message formats that you

want to deploy with the CICS Web services infrastructure. CICS allows you to do this, but you will not be able to use some of the functions that CICS provides, such as the Web services assistant, and the SOAP message handlers.

Do you intend to use the CICS Web services assistant to generate the mappings between your data structures and SOAP messages?

The assistant provides a rapid deployment of many applications into a Web services setting with little or no additional programming. And when additional programming is required, it is usually straightforward, and can be done without changing existing business logic.

However, there are cases which are better handled without using the Web services assistant. For example, if you have existing code that maps data structures to SOAP messages, there is no advantage in reengineering your application with the Web services assistant.

Do you intend to use an existing service description, or create a new one?

In some situations, you will be obliged to use an existing service description as a starting point. For example:

- Your application is a service requester, and it is designed to invoke an existing Web service.
- Your application is a service provider, and you want it to conform to an existing industry-standard service description.

In other situations, you may need to create a new service description for your application.

Next steps:

- Planning a service provider
- Planning a service requester

Planning a service provider application

In general, CICS applications should be structured to ensure separation of business logic and communications logic. Following this practice will help you to deploy new and existing applications in a Web service provider in a straightforward way. You will, in some situations, need to interpose a simple wrapper program between your application program and CICS Web service support.

Figure 15 shows a typical application which is partitioned to ensure a separation between communication logic and business logic.

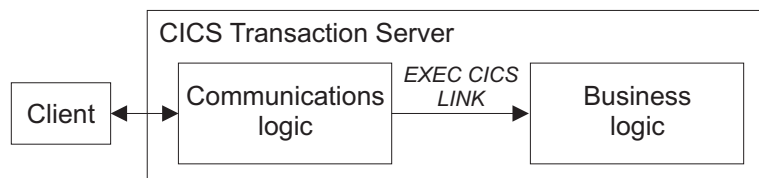


Figure 15. Application partitioned into communications and business logic

In many cases, you can deploy the business logic directly as a service provider application. This is illustrated in Figure 16 on page 39.

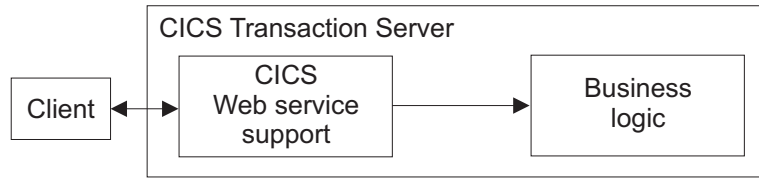


Figure 16. Simple deployment of CICS application as a Web service provider

To use this simple model, the following conditions apply:

When you are using the CICS Web services assistant to generate the mapping between SOAP messages and application data structures:

The data types used in the interface to the business logic must be supported by the CICS Web services assistant. If this is not the case, you must interpose a wrapper program between CICS Web service support and your business logic.

You will also need a wrapper program when you deploy an existing program to provide a service that conforms to an existing Web service description: if you process the Web service description using the assistant, the resulting data structures are very unlikely to match the interface to your business logic.

When you are not using the CICS Web services assistant:

Message handlers in your service provider pipeline must interact directly with your business logic.

Using a wrapper program

Use a wrapper program when the CICS Web services assistant cannot generate code to interact directly with the business logic. For example, the interface to the business logic might use a data structure which the CICS Web services assistant cannot map directly into a SOAP message. In this situation, you can use a wrapper program to provide any additional data manipulation that is required:

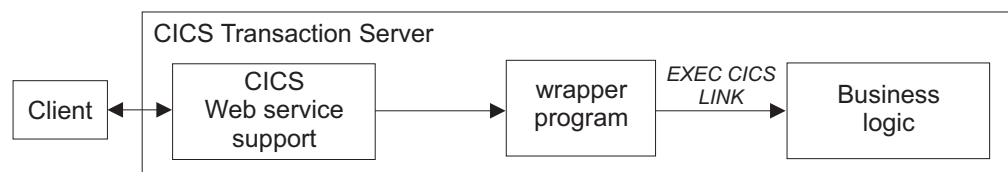


Figure 17. Deployment of CICS application as a Web service provider using a wrapper program

You will need to design a second data structure that the assistant can support, and use this as the interface to your wrapper program. The wrapper program then has two simple functions to perform:

- move data between the two data structures
- invoke the business logic using its existing interface

Error handling

If you are planning to use the CICS Web services assistant, you should also consider how to handle rolling back changes when errors occur. When a SOAP request message is received from a service requester, the SOAP message is transformed by CICS just before it is passed to your application program. If an error

occurs during this transformation, CICS does not automatically roll back any work that has been performed on the message. For example, if you plan to add some additional processing on the SOAP message using handlers in the pipeline, you need to decide if they should roll back any recoverable changes that they have already performed.

On outbound SOAP messages, for example when your service provider application program is sending a response message to a service requester, if CICS encounters an error when generating the response SOAP message, all of the recoverable changes made by the application program are automatically backed out. You should consider whether adding synchronization points is appropriate for your application program.

If you are planning to use Web service atomic transactions in your provider application, and the Web service requester also supports atomic transactions, any error that causes CICS to roll back a transaction would also cause the remote requester to roll back its changes.

Planning a service requester application

In general, CICS applications should be structured to ensure separation of business logic and communications logic. Following this practice will help you to deploy new and existing applications in a Web service requester in a straightforward way. You will, in almost every situation, need to interpose a simple wrapper program between your application program and CICS Web service support.

Figure 18 shows a typical application which is partitioned to ensure a separation between communication logic and business logic. The application is ideally structured for reuse of the business logic in a Web service requester.

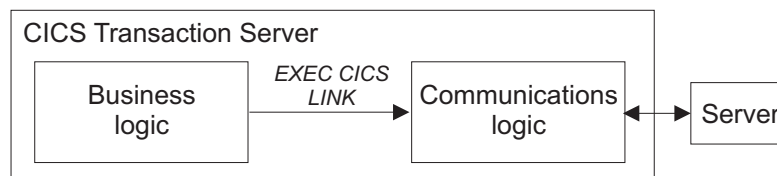


Figure 18. Application partitioned into communications and business logic

You cannot use the existing EXEC CICS LINK command to invoke CICS Web services support in this situation:

- When you are using the CICS Web services assistant to generate the mapping between SOAP messages and application data structures, you must use an EXEC CICS INVOKE WEBSERVICE command, and pass the application's data structure to CICS Web services support. Also, the data types used in the interface to the business logic must be supported by the CICS Web services assistant.

However, if the target WEBSERVICE that your application program invokes is provider mode, i.e. a value has been defined for the PROGRAM attribute, CICS automatically optimizes the request using the EXEC CICS LINK command.

- When you are not using the CICS Web services assistant, you must construct your own messages, and link to program DFHPIRT.

Either way, it follows that your business logic cannot invoke a Web service directly unless you are prepared to change the program. For the Web services assistant,

this option is shown in Figure 19, but it is not advisable in either case.

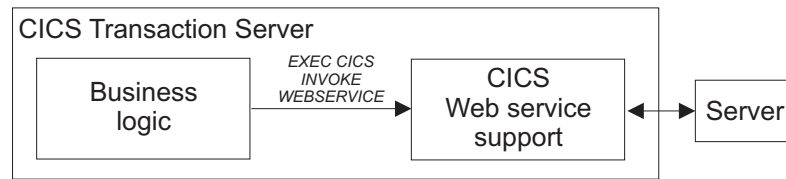


Figure 19. Simple deployment of CICS application as a Web service requester

Using a wrapper program

A better solution, which keeps the business logic almost unchanged, is to use a wrapper program. The wrapper, in this case, has two purposes:

- It issues an EXEC CICS INVOKE WEBSERVICE command, or an EXEC CICS LINK PROGRAM(DFHPIRT), on behalf of the business logic. The only change in the business logic is the name of the program to which it links.
- It can, if necessary, provide any data manipulation that is required if your application uses a data structure which the CICS Web services assistant cannot map directly into a SOAP message.

For the case when the Web services assistant is used, this structure is illustrated in Figure 20.

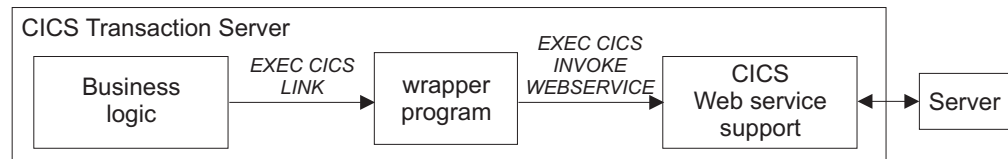


Figure 20. Deployment of CICS application as a Web service requester using a wrapper program

Error handling

If you are planning to use the CICS Web services assistant, you should also consider how to handle rolling back changes when errors occur. If your service requester application receives a SOAP fault message from the service provider, you need to decide how your application program should handle the fault message. CICS does not automatically roll back any changes when a SOAP fault message is received.

If you are planning to implement Web service atomic transactions in your requester application program, the error handling is different. If the remote service provider encounters an error and rolls back its changes, a SOAP fault message is returned and the local transaction in CICS also rolls back. If local optimization is in effect, the service requester and provider use the same transaction. If the provider encounters an error, any changes made by the transaction in the requester are also rolled back.

Migrating from the SOAP for CICS feature

If you use the SOAP for CICS feature, you must perform a number of tasks to migrate applications that use the feature.

The support for Web services provided in CICS Transaction Server is substantially different from that provided in the feature. The SOAP for CICS feature relies to a considerable extent upon user-written code, and therefore it is not possible to set out a step-by-step migration task. However, here are some of the things you will need to think about.

- Consider using the Web services assistant to construct and parse SOAP messages. If you decide to do so, you are advised to discard your existing message adapters, and design new wrapper programs to replace them, as it is unlikely that you will be able to reuse significant amounts of code in your adapters.
- If you use SOAP messages, but decide not to use the Web services assistant, you may be able to reuse your existing code for constructing and parsing the messages. However, you should consider whether to use the CICS-provided SOAP message handlers, because they are designed to work with SOAP 1.1 and SOAP 1.2 messages.
- Review your use of containers. The SOAP for CICS feature uses BTS containers, whereas CICS Transaction Server uses channel containers. You will need to review your programs and change any BTS-related commands required by the feature. You will also need to review the name and usage of each container, as most of these have changed.
- Consider how to migrate the function that was provided by your pipeline programs. The pipeline in the SOAP for CICS feature has a fixed number of user-written programs, each with a designated purpose. The function provided by some of these programs is provided in CICS Transaction Server by the CICS-provided SOAP message handlers, so you may be able to dispense with these programs altogether.

On the other hand, CICS Transaction Server lets you define as many programs in your pipeline as you need. Therefore, you should consider whether the function performed by your pipeline programs should be restructured to take advantage of the new framework.

In any case, the way that pipeline programs communicate with CICS, and with one another, has changed, so you will need to review these programs to see if they can be reused in the new environment.

In the SOAP for CICS feature, you could have just one pipeline for all your service provider applications, and one for all your service requesters. In CICS Transaction Server, you can configure many different pipelines. Therefore, it is possible that the logic you provided in your pipeline programs to distinguish one application from another can be replaced by CICS resource definitions. For example, in a service provider, code that distinguishes between applications based upon a URI, can be replaced with a suitable set of URIMAP resources

Planning to use Web services

Before you can plan to use Web services in CICS, you need to consider these questions for each application:

Do you plan to deploy your CICS application in the role of a service provider or a service requester?

You may have a pair of applications that you want to connect using CICS support for Web services. In this case, one application will be the service provider; the other will be the service requester.

Do you plan to use your existing application programs, or write new ones?

If your existing applications are designed with a well defined interface to the

business logic, you will probably be able to use them in a Web services setting, either as a service provider or a service requester. However, in most cases, you will need to write a wrapper program that connects your business logic to the Web services logic.

If you plan to write new applications, you should aim to keep your business logic separated from your Web services logic, and, once again, you will need to write a wrapper program to provide this separation. However, if your application is designed with Web services in mind, the wrapper may prove to be simpler to write.

Do you intend to use SOAP messages?

SOAP is fundamental to the Web services architecture, and much of the support that is provided in CICS assumes that you will use SOAP. However, there may be situations where you wish to use other message formats. For example, you may have developed your own message formats that you want to deploy with the CICS Web services infrastructure. CICS allows you to do this, but you will not be able to use some of the functions that CICS provides, such as the Web services assistant, and the SOAP message handlers.

Do you intend to use the CICS Web services assistant to generate the mappings between your data structures and SOAP messages?

The assistant provides a rapid deployment of many applications into a Web services setting with little or no additional programming. And when additional programming is required, it is usually straightforward, and can be done without changing existing business logic.

However, there are cases which are better handled without using the Web services assistant. For example, if you have existing code that maps data structures to SOAP messages, there is no advantage in reengineering your application with the Web services assistant.

Do you intend to use an existing service description, or create a new one?

In some situations, you will be obliged to use an existing service description as a starting point. For example:

- Your application is a service requester, and it is designed to invoke an existing Web service.
- Your application is a service provider, and you want it to conform to an existing industry-standard service description.

In other situations, you may need to create a new service description for your application.

Next steps:

- Planning a service provider
- Planning a service requester

The CICS catalog manager example application

The CICS catalog example application is a working COBOL application that is designed to illustrate best practice when connecting CICS applications to external clients and servers.

The example is constructed around a simple sales catalog and order processing application, in which the end user can perform these functions:

- List the items in a catalog.

- Inquire on individual items in the catalog.
- Order items from the catalog.

The catalog is implemented as a VSAM file.

The base application has a 3270 user interface, but the modular structure, with well-defined interfaces between the components, makes it possible to add further components. In particular, the application comes with Web service support, which is designed to illustrate how you can extend an existing application into the Web services environment.

Chapter 6. Configuring your CICS system for Web services

Before you can use Web services, your CICS system must be correctly configured.

1. Ensure that you have installed Language Environment® support for PL/I. For more information, see the *CICS Installation Guide*.
2. Activate z/OS Support for Unicode. You must enable the z/OS conversion services and install a conversion image which specifies the conversions that you want CICS to perform. For more information, see *z/OS Support for Unicode: Using Conversion Services*.

Related tasks

dfha1n4.htm#dfha1lm

CICS resources for Web services

The following CICS resources support Web services in CICS:

PIPELINE

A PIPELINE resource definition is required in every case. It provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE definition defines an infrastructure that can be used by many applications. The information about the message handlers is supplied indirectly: the PIPELINE specifies the name of an HFS file which contains an XML description of the handlers and their configuration.

A PIPELINE resource that is created for a service requester cannot be used for a service provider, and vice versa. The two sorts of PIPELINE are distinguished by the contents of the pipeline configuration file that is specified in the CONFIGFILE attribute: for a service provider, the top level element is <provider_pipeline>; for a service requester it is <requester_pipeline>.

WEBSERVICE

A WEBSERVICE resource definition is required only when the mapping between application data structure and SOAP messages has been generated using the CICS Web services assistant. It defines aspects of the run time environment for a CICS application program deployed in a Web services setting.

Although CICS provides the usual resource definition mechanisms for WEBSERVICE resources, they are typically created automatically from a Web service binding file when the PIPELINE's pickup directory is scanned. This happens when the PIPELINE resource is installed, or as a result of a PERFORM PIPELINE SCAN command. The attributes applied to the WEBSERVICE resource in this case come from a Web services binding file, which is created by the Web services assistant; information in the binding file comes from the Web service description, or is supplied as a parameter of the Web services assistant.

A WEBSERVICE resource that is created for a service requester cannot be used for a service provider, and vice versa. The two sorts of WEBSERVICE are distinguished by the PROGRAM attribute: for a service provider, the attribute must be specified; for a service requester it must be omitted.

URIMAP

A URIMAP definition is required only in a service provider, and contains

information that maps the URI of an inbound Web service request to the other resources (such as the PIPELINE) that will service the request.

Although CICS provides the usual resource definition mechanisms, for service providers deployed using the CICS Web services assistant the URIMAP resources are typically created automatically from a Web service binding file when the PIPELINE's pickup directory is scanned. This happens when the PIPELINE resource is installed, or as a result of a PERFORM PIPELINE SCAN command. The attributes applied to the URIMAP resource in this case come from a Web services binding file, which is created by the Web services assistant; information in the binding file comes from the Web service description, or is supplied as a parameter of the Web services assistant.

TCPIPSERVICE

A TCPIPSERVICE definition is required in a service provider that uses the HTTP transport, and contains information about the port on which inbound requests are received.

The resources that are required to support a particular application program depends upon the following:

- Whether the application program is a service provide or a service requester.
- Whether the application is deployed with the CICS Web services assistant.

Service requester or provider	CICS Web services assistant used	PIPELINE required	WEBSERVICE required	URIMAP required	TCPIPSERVICE required
Provider	Yes	Yes	Yes (but see note 1)	Yes (but see note 1)	See note 2
	No	Yes	No	Yes	See note 2
Requester	Yes	Yes	Yes	No	No
	No	Yes	No	No	No

Notes:

1. When the CICS Web service assistant is used to deploy an application program, the WEBSERVICE and URIMAP resources can be created automatically when the PIPELINE's pickup directory is scanned. This happens when the PIPELINE resource is installed, or as a result of a PERFORM PIPELINE SCAN command.
2. A TCPIPSERVICE resource is required when the HTTP transport is used. When the WebSphere® MQ transport is used, a TCPIPSERVICE resource is not required.

Typically, when you deploy many Web services applications are deployed in a CICS system, you will have more than one of each type of resource. In this case, you can share some resources between applications.

For each ...	You can have ...
Pipeline configuration file	<ul style="list-style-type: none"> • More than one PIPELINE resource that refers to the file
PIPELINE resource	<ul style="list-style-type: none"> • More than one URIMAP resource that refers to the PIPELINE • More than one WEBSERVICE resource that refers to the PIPELINE • More than one Web service binding file in the PIPELINE's pickup directory

For each ...	You can have ...
Web service binding file	<ul style="list-style-type: none"> • Just one URIMAP resource that is automatically generated from the binding file. But you can define further URIMAPs using RDO. • Just one WEBSERVICE resource that is automatically generated from the binding file. But you can define further WEBSERVICES using RDO.
WEBSERVICE	<ul style="list-style-type: none"> • More than one URIMAP resource. If the WEBSERVICE resource is automatically generated from the binding file, there is just one corresponding URIMAP resource. But you can define further URIMAP resources using RDO.
URIMAP	<ul style="list-style-type: none"> • Just one TCPIPService when it is explicitly named in the URIMAP resource.
TCPIPService	<ul style="list-style-type: none"> • Many URIMAP resources.

Configuring CICS to use the WebSphere MQ transport

If you want to use the WebSphere MQ transport with Web services in CICS, you must configure your CICS region accordingly.

You can find more detailed information in the *MQ Series System Setup Guide*.

1. Include the following libraries in the STEPLIB concatenation.

```
thlqual.SCSQANLx
thlqual.SCSQAUTH
```

where:

thlqual is the high level qualifier for the MQ libraries.
x is the language letter for national language.

2. Include the following libraries in the DFHRPL concatenation.

```
thlqual.SCSQLOAD
thlqual.SCSQANLx
thlqual.SCSQCICS
thlqual.SCSQAUTH
```

thlqual is the high level qualifier for the MQ libraries.
x is the language letter for national language.

3. Specify the following CICS system initialization parameters.

```
INITPARM=(CSQCPARM='SN=queuemanager,TN=traceptid,IQ=initiation_queue')
MQCONN=YES
```

where:

queuemanager is the subsystem name.
traceptid is the trace number that identifies the adapter in CICS trace entries.
initiation_queue is the name of the default initiation queue.

4. Ensure that the coded character set identifiers (CCSIDs) used by your queue manager and by CICS, and the UTF-8 and UTF-16 code pages are configured to z/OS conversion services. The CICS code page is specified in the **LOCALCCSID** system initialization parameter.
5. Install the MQ CSD group CSQCAT1 in the CICS region.

The WebSphere MQ transport

CICS can receive and send SOAP messages to WebSphere MQ (WMQ) using the WMQ transport, both in the role of service provider and service requester.

As a **service provider**, CICS uses WMQ triggering to process SOAP messages from an application queue. Triggering works by using an initiation queue and local queues. A local (application) queue definition includes:

- the criteria for when a trigger message should be generated. For example, when the first message arrives on the local queue, or for every message that arrives on the local queue. For CICS SOAP processing, you should specify that triggering occurs when the first message arrives on the local queue.

The local queue definition can also specify that trigger data should be passed to the target application, and in the case of CICS SOAP processing (transaction CPIL), this specifies the default target URL to be used if this is not passed with the inbound message.

- the *Process name* that identifies the *Process definition*. The Process definition describes how the message should be processed. In the case of CICS SOAP processing, this should specify the CPIL transaction.
- the name of the initiation queue that the trigger message should be sent to.

When a message arrives on the local queue, the Queue Manager generates and sends a trigger message to the specified initiation queue. The trigger message includes the information from the process definition. The trigger monitor retrieves the trigger message from the initiation queue and schedules the CPIL transaction to start processing the messages on the local queue. For more information about triggering, see the *WebSphere MQ Application Guide*.

You can configure CICS, so that when a message arrives on a local queue, the trigger monitor (provided by WMQ) schedules the CPIL transaction to process the messages on the local queue and drive the CICS SOAP pipeline to process the SOAP messages on the queue.

When CICS constructs a response to a SOAP message that is received from Websphere MQ, the correlation id field is populated with the message id of the input message, unless the report option MQRO_PASS_CORREL_ID has been set. If this report option has been set, the correlation id is propagated from the input message to the response.

As a **service requester**, on outbound requests you can specify that the responses for the target Web service should be returned on a particular reply queue.

In both cases, CICS and WMQ require configuration to define the necessary resources and queues.

Defining local queues in a service provider

To use the WebSphere MQ transport in a service provider, you must define an initiation queue, one or more local queues that store input messages until they are processed, and one trigger process that specifies the CICS transaction that will process the input messages.

1. Define an initiation queue. Use the following command:

```
DEFINE
QLOCAL('initiation_queue')
DESCR('description')
```

where *initiation_queue* is the same as the value specified in IQ= in CSQCPARM in the **INITPARM** system initialization parameter.

2. For each local input queue, define a QLOCAL object. Use the following command:

```
DEFINE
QLOCAL('queuename')
DESCR('description')
PROCESS(processname)
INITQ('initiation_queue')
TRIGGER
TRIGTYPE(FIRST)
TRIGDATA('default_target_service')
BOTHRESH(nnn)
BOQNAME('requeuename')
```

where:

queuename is the local queue name.

processname is the name of the process instance that identifies the application started by the queue manager when a trigger event occurs. Specify the same name on each QLOCAL object.

initiation_queue is the name of the initiation queue to be used that you specified in the previous step.

default_target_service is the default target service to be used if a service is not specified on the request. The target service is of the form '/string' and is used to match the path of a URIMAP definition. for example '/SOAP/test/test1'. Note that the first character must be '/' .

nnn is the number of retries that will be attempted.

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

3. Define a PROCESS object that specifies the trigger process. Use the following command:

```
DEFINE
PROCESS(processname)
APPLTYPE(CICS)
APPLICID(CPIL)
```

where:

processname is the name of the process, and must be the same as the name that is used when defining the input queues.

Defining local queues in a service requester

When you use the MQ transport for outbound requests in a service requester, you can specify in the URI for the target Web service that your responses should be returned on a predefined reply queue. If you do so, you must define each reply queue with a QLOCAL object.

If the URI associated with a request does not specify a reply queue, CICS will use a dynamic queue for the reply.

Optional: To define each QLOCAL object that specifies a predefined reply queue, use the following command.

```
DEFINE
QLOCAL('reply_queue')
DESCR('description')
BOTHRESH(nnn)
```

where:

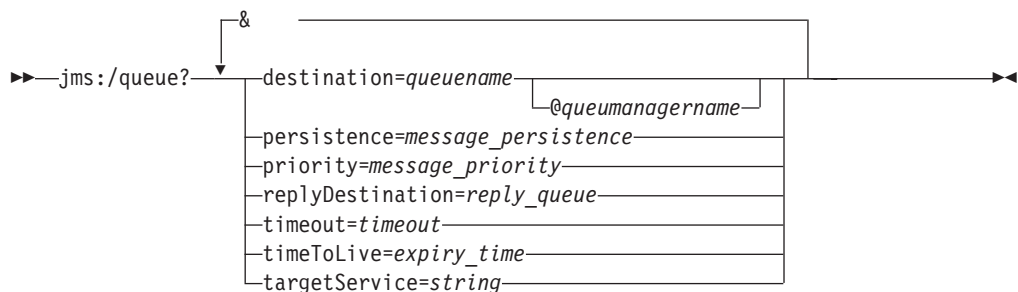
reply_queue is the local queue name.

nnn is the number of retries that will be attempted.

The URI for the MQ transport

When communication between the service requester and service provider uses MQ, the URI of the target is in a form that identifies the target as a queue, and includes information to specify how the request and response should be handled by MQ.

Syntax



CICS uses the following options; other Web service providers may use further options that are not described here. CICS ignores any options that it does not support and that are coded in the URI. However, the entire URI is passed to the service provider. CICS is not sensitive to the case of the option names. However, some other implementations that support this style of URI are case-sensitive.

destination=queueename [*@queuanagername*]

queueename is the name of the input queue in the target queue manager

queuanagername is the name of the target queue manager

persistence=message_persistence

Specify one of the following:

- 0** Messages are not persistent.
- 1** Messages are persistent.

2 Persistence is defined by the default queue persistence.

If the option is not specified or is specified incorrectly, the default queue persistence is used.

priority=*message_priority*

Specifies the message priority as an integer in the range 0 to 99999999.

replyDestination=*reply_queue*

Specifies the queue to be used for the response message. If this option is not specified, CICS will use a dynamic queue for the response message. You must define the reply queue in a QLOCAL object before using this option.

timeout=*timeout*

The timeout in milliseconds for which the service requester will wait for a response. If a value of zero is specified, or if this option is omitted, the request will not time out.

timeToLive=*expiry-time*

Specifies the expiry time for the request in milliseconds. If the option is not specified or is specified incorrectly, the request will not expire.

targetService=*string*

Identifies the target service. If CICS is the service provider, then the target service should be of the form '/string', as CICS will use this as the path when attempting to match with URIMAP. If not specified, the value specified in TRIGDATA on the input queue at the service provider is used.

This is an example of a URI for the MQ transport:

```
jms:/queue?destination=queue01@cics007&timeToLive=10&replyDestination=rqueue05&targetService=/myservice
```

Configuring CICS to support persistent messages

CICS provides support for sending persistent messages using the WMQ transport protocol to a Web service provider application that is deployed in a CICS region.

CICS uses Business Transaction Services (BTS) to ensure that persistent messages are recovered in the event of a CICS system failure. For this to work correctly, follows these steps:

1. Use IDCAMS to define the local request queue and repository file to MVS. You must specify a suitable value for STRINGS for the file definition. The default value of 1 is unlikely to be sufficient, and you are recommended to use 10 instead.
2. Define the local request queue and repository file to CICS. Details of how to define the local request queue to CICS are described in “Defining local queues in a service provider” on page 49. You must specify a suitable value for STRINGS in the file definition. The default value of 1 is unlikely to be sufficient, and it is recommended that you use 10 instead.
3. Define a PROCESSTYPE resource with the name DFHMQSOA, using the repository file name as the value for the FILE option.

For one way request messages, if the Web service abends or backs out, sufficient information is retained to allow a transaction or program to retry the failing request, or to report the failure appropriately. You need to provide this recovery transaction or program. See “Persistent message processing” on page 52 for details.

Persistent message processing

When a Web service request is received in a WMQ persistent message, CICS creates a unique BTS process with the process type DFHMQSOA. Data relating to the inbound request is captured in BTS data-containers that are associated with the process.

The process is then scheduled to run asynchronously. If the Web service completes successfully and commits, CICS deletes the BTS process. This includes the case when a SOAP fault is generated and returned to the Web service requester.

Error processing

If an error occurs when creating the required BTS process, the Web service transaction abends, and the inbound Web service request is not processed. If BTS is not usable, message DFHPI0117 is issued, and CICS continues without BTS, using the existing channel-based container mechanism.

If a CICS failure occurs before the Web service starts or completes processing, BTS recovery ensures that the process is rescheduled when CICS is restarted.

If the Web service abends and backs out, the BTS process is marked complete with an ABENDED status. For request messages that require a response, a SOAP fault is returned to the Web service requester. The BTS process is cancelled, and CICS retains no information about the failed request. CICS issues message DFHBA0104 on transient data queue CSBA, and message DFHPI0117 on transient data queue CPIO.

For one way messages, there is no way to return information about the failure to the requester so the BTS process is retained in a COMPLETE ABENDED state. CICS issues message DFHBA0104 on transient data queue CSBA, and DFHPI0116 on transient data queue CPIO.

You can use the CBAM transaction to display any COMPLETE ABENDED processes, or you can supply a recovery transaction to check for COMPLETE ABENDED processes of the DFHMQSOA and take appropriate action.

For example, your recovery transaction could:

1. Reset the BTS process using the RESET ACQPROCESS command.
2. Issue the RUN ASYNC command to retry the failing Web service. It could keep a retry count in another data-container on the process, to avoid repeated failure.
3. Use information in the associated data-containers to report on the problem:

The DFHMQORIGINALMSG data-container contains the message received from WMQ, which might contain RFH2 headers.

The DFHMQMSG data-container contains the WMQ message with any RFH2 headers removed.

The DFHMQDLQ data-container contains the name of the dead letter queue associated with the original message.

The DFHMQCONT data-container contains the WMQ MQMD control block relating to the MQ GET for the original message.

Chapter 7. The CICS Web services assistant

The CICS Web services assistant is a set of batch utilities which can help you to transform existing CICS applications into Web services and to enable CICS applications to use Web services provided by external providers. The assistant supports rapid deployment of CICS applications for use in service providers and service requesters, with the minimum of programming effort.

When you use the Web services assistant for CICS, you do not have to write your own code for parsing inbound messages and for constructing outbound messages; CICS maps data between the body of a SOAP message and the application program's data structure.

Resource definitions are, for the most part, generated and installed automatically. You do have to define PIPELINE resources, but you can, in many cases, use one of the pipeline configuration files that CICS provides. These are:

basicsoap11provider.xml

Pipeline configuration file for a service provider using the SOAP 1.1 message handler.

basicsoap11requester.xml

Pipeline configuration file for a service requester using the SOAP 1.1 message handler.

The assistant can create a WSDL document from a simple language structure, or a language structure from an existing WSDL document, and supports COBOL, C/C++, and PL/I. It also generates information used to enable automatic runtime conversion of the SOAP messages to containers and COMMAREAs, and *vice versa*.

However, the assistant cannot deal with every possibility, and there are times when you will need to take a different approach. For example:

You don't want to use SOAP messages

If you prefer to use a non-SOAP protocol for your messages, you can do so. However, your application programs will be responsible for parsing inbound messages, and constructing outbound messages.

You want to use SOAP messages, but don't want CICS to parse them

For an inbound message, the assistant maps the SOAP body to an application data structure. In some applications, you may want to parse the SOAP body yourself.

The CICS Web services assistant does not support your application's data structure

Although the CICS Web services assistant supports the most common data types and structures, there are some which are not supported. In this situation, you should first consider providing a program layer that maps your application's data to a format that the assistant can support. If this is not possible, you will need to parse the message yourself.

If you decide not to use the CICS Web services assistant, you will have to:

- Provide your own code for parsing inbound messages, and constructing outbound messages
- Provide your own pipeline configuration file
- Define and install your own URIMAP and PIPELINE resources

The CICS Web services assistant comprises two utility programs:

DFHLS2WS

Generates a Web service binding file from a language structure. This utility also generates a Web service description.

DFHWS2LS

Generates a Web service binding file from a Web service description. This utility also generates a language structure that you can use in your application programs.

The JCL procedures to run both programs are in the *hlq.XDFHINST* library.

Using the CICS Web services assistant to deploy a service provider application

The CICS Web services assistant simplifies the task of deploying your CICS applications in a service provider setting.

When you use the assistant to deploy a CICS application as a service provider, you have two options:

- Start with a Web service description, and use the assistant to generate the language data structures.

Use this option when you are implementing a service provider that conforms with an existing Web service description.

- Start with the language data structures, and use the assistant to generate the Web service description.

Use this option when you are exposing an existing program as a Web service, and are willing to expose aspects of the program's interfaces in the Web service description and the SOAP messages.

Creating a service provider application from a Web service description

Using the CICS Web services assistant, you can create a service provider application from a Web service description.

Your Web services description must be in a file in HFS.

1. Generate a Web service binding file. Use batch program DFHWS2LS to generate a Web service binding file. As well as the Web service binding file, the program generates a language data structure.
2. Copy the Web service binding file to the pickup directory of the PIPELINE resource that you want to use for your Web service application.
3. Use the language data structure generated in 1 to write your wrapper program. The wrapper program will manipulate the data into the correct form to interact with the business logic.
4. If you do not have a suitable PIPELINE resource definition in your system, create and install one. The PIPELINE resource specifies the XML file which defines the message handlers which will be used to process the inbound request and the reply. Typically, many applications can use the same PIPELINE definition, and if you already have a suitable PIPELINE in your system, you do not need to perform this step.
5. Create and install a URIMAP which matches the URI used to invoke the Web service. The URIMAP specifies the names of the WEBSERVICE resource, and of the PIPELINE resource that provide further details of how the Web service request is processed.

The URIMAP can be created automatically using the scan mechanism. In this case, CICS gets the information needed to build the URIMAP from the Web service binding file.

6. Create and install a WEBSERVICE which specifies the location of the WSDL and of the WSBIND file. Although you can use RDO to create a WEBSERVICE, the recommended method is to scan for WSBIND files. This creates WEBSERVICE definitions which are consistent with the WSDL.

Creating a service provider application from a data structure

Using the CICS Web services assistant, you can create a service provider application from a high level language data structure.

Before you can process your high level language data structures, you must ensure that:

- The data structures are defined separately from the source program (for example in a COBOL copy book).
- If your PL/I or COBOL application program uses different data structures for its input and its output, the data structures are defined in two different members in a partitioned data set. If the same structure is used for input and output, the structure should be defined in a single member.

For C and C++, your data structures can be in the same member in a partitioned data set.

Which data structures you process depend upon whether you are using a wrapper program:

- If you are using a wrapper program, the copy book is the interface to the wrapper program.
 - If you are not using a wrapper program, the copy book is the interface to the business logic.
1. Generate a Web service binding file. Use batch program DFHLS2WS to generate a Web service binding file. As well as the Web service binding file, the program generates Web service description. The service description can be used when building a service requester that interacts with your service. You will also need the Web service description if you intend to perform run time validation of SOAP messages for the service.
 2. Copy the Web service binding file to the pickup directory of the PIPELINE resource that you want to use for your Web service application.
 3. If you do not have a suitable PIPELINE resource definition in your system, create and install one. The PIPELINE resource specifies the XML file which defines the message handlers which will be used to process the inbound request and the reply. Typically, many applications can use the same PIPELINE definition, and if you already have a suitable PIPELINE in your system, you do not need to perform this step.
 4. Create and install a URIMAP which matches the URI used to invoke the Web service. The URIMAP specifies the names of the WEBSERVICE resource, and of the PIPELINE resource that provide further details of how the Web service request is processed.

The URIMAP can be created automatically using the scan mechanism. In this case, CICS gets the information needed to build the URIMAP from the Web service binding file.

5. Create and install a WEBSERVICE which specifies the location of the WSDL and of the WSBIND file. Although you can use RDO to create a WEBSERVICE,

the recommended method is to scan for WSBIND files. This creates WEBSERVICE definitions which are consistent with the WSDL.

You should make the Web services description available to anyone who needs to develop a service requester that will access your service.

Customizing generated Web service description documents

The Web service description (WSDL) documents that are generated by DFHLS2WS contain some automatically generated content that it might be appropriate for you to change before publishing. For example, the WSDL generated by DFHLS2WS assumes the use of SOAP version 1.1 over HTTP.

Customizing WSDL documents can result in regenerating the Web services binding file and in some cases, writing a wrapper program.

1. If you want to advertise support for SOAP 1.2, HTTPS or communicate using WebSphere MQ, then you need to change the `wsdl:service` and `wsdl:binding` tags at the end of the WSDL document. The generated WSDL includes comments to assist you in making these changes. Changing these elements does not require you to regenerate the Web services binding file.
2. If you want to supply the network location of your Web service, add the details to the `soap:address` within the `wsdl:service` element.
 - a. If you are using an HTTP-based protocol, replace *my-server* with the TCP/IP host name of your CICS region and *my-port* with the port number of the TCPIP SERVICE resource.
 - b. If you are using WebSphere MQ as the transport protocol, replace *myQueue* with the name of the appropriate queue.

These changes can be made without requiring any change to the Web services binding file.

3. Consider if the automatically generated names in the WSDL document are appropriate for your purposes. The values that you can rename are:
 - The `targetNamespace` of the WSDL document
 - The `targetNamespace` of the XML schemas within the WSDL document
 - The `wsdl:portType` name
 - The `wsdl:operation` name
 - The `wsdl:binding` name
 - The `wsdl:service` name
 - The names of the fields in the XML schemas within the WSDL document.

These values form part of the programmatic interface to which a client program must be coded. If the generated names are not sufficiently meaningful, it could make maintenance of your application code harder over a long period of time. It is recommended that you use the DFHLS2WS parameters **REQUEST-NAMESPACE** and **RESPONSE-NAMESPACE** to change the `targetNamespace` of the XML schemas.

If you change any of these values, you need to regenerate the Web services binding file using DFHWS2LS. The language structures that are produced will not be the same as your existing language structure, but are compatible with your existing application, so no application changes are required. However, you can ignore the new language structures and use the new Web services binding file with the original structures.

4. Consider if the COMMAREA fields exposed in the XML schemas are appropriate. You might want to consider removing those fields that are not helpful to a Web service client developer. For example:
 - fields that are only used for output values could be removed from the schema that maps the input data structures
 - filler fields
 - automatically generated annotations

If you make any of these changes, you need to regenerate the Web services binding file using DFHWS2LS. The new language structures that are generated are not compatible with the original language structures, so you need to write a wrapper program to map data from the new representation to the old one. This wrapper program needs to LINK to the target application program and then map the returned data.

This level of customization requires the most effort, but results in the most meaningful programmatic interfaces for your Web services client developers to work with.

5. If you want to put the generated WSDL document through DFHWS2LS to create new language structures, decide whether to keep the annotations in the WSDL document. The annotations override the normal mapping rules when DFHWS2LS generates the language structures. Overriding the mapping rules ensures that the generated language structures are compatible with the version that was used by DFHLS2WS. If you want to use the default mapping rules to produce the language structures, remove the annotations.

For an example of a WSDL document, see “An example of the generated WSDL document” on page 225.

Using the CICS Web services assistant to deploy a service requester application

The CICS Web services assistant simplifies the task of deploying your CICS applications in a service requester setting.

When you use the CICS Web services assistant to deploy a CICS application as a service requester, you must start with a Web service description, and generate the language data structures from it.

The alternative of starting with the language data structures, and generating the Web service description is not recommended for a service requester: it is normally the service provider's responsibility to supply a description of the service.

Creating a service requester application from a Web service description

Using the CICS Web services assistant, you can create a service requester application from a Web service description.

Your Web services description must be in a file in HFS.

1. Generate a Web service binding file. Use batch program DFHWS2LS to generate a Web service binding file.

Important: Do not specify the PROGRAM parameter when you use DFHWS2LS. This parameter applies only to a service provider. As well as the Web service binding file, the program generates a language data structure.

2. Copy the Web service binding file to the pickup directory of the PIPELINE resource that you want to use for your Web service application. Make sure that the PIPELINE is configured for a service requester - that is, the top level element of the configuration file specified in the CONFIGFILE attribute is <requester_pipeline>.
3. Use the language data structure generated in 1 on page 57 to write your wrapper program. Use an EXEC CICS INVOKE WEBSERVICE command in your wrapper program to communicate with the Web service. The options on the command include:
 - The name of the WEBSERVICE resource
 - The operation for which the Web service is being invoked
4. If you do not have a suitable pipeline configuration file, create one. Typically, many service requester applications can use the same pipeline configuration, and if you already have a suitable configuration file, you do not need to perform this step.
5. If you do not have a suitable PIPELINE resource definition in your system, create and install one. The PIPELINE resource specifies the name of the pipeline configuration file. Typically, many service requester applications can use the same PIPELINE definition, and if you already have a suitable PIPELINE in your system, you do not need to perform this step.
6. Create and install a WEBSERVICE resource definition. Although you can use RDO to create a WEBSERVICE, the recommended method is to scan for WSBIND files. Use the PERFORM PIPELINE SCAN command. This creates WEBSERVICE definitions which are consistent with the WSDL.
7. Write a wrapper program that you can substitute for your communications logic.

Creating the CICS infrastructure for a service provider

To create the CICS infrastructure for a service provider, you must define and install a number of CICS resources. In many cases, CICS can generate some of these resources automatically.

For a service provider application deployed with the help of the CICS Web services assistant, you will need to define the following:

The transport infrastructure

If you are using the MQ transport, you must define one or more local queues that store input messages until they are processed, and one trigger process that specifies the CICS transaction that will process the input messages.

If you are using the HTTP transport, you must define a TCPIPSERVICE that contains information about defines the port on which inbound requests are received.

A PIPELINE resource definition

With its associated *pipeline configuration file*, the PIPELINE resource defines the attributes of the pipeline which is used to process inbound Web service requests, and the responses. Typically, one pipeline can process

requests for many different Web services, and when you deploy a new Web service in your CICS system, you will be able to use an existing pipeline.

As well as the configuration file, the PIPELINE resource specifies a *pickup directory*, which contains *Web service binding files*.

When you install a PIPELINE resource, or when you issue a PERFORM PIPELINE SCAN command (using CEMT or the CICS system programming interface), CICS reads the files in the pickup directory, and creates URIMAP and WEBSERVICE resources dynamically.

PROGRAM resource definitions

Unless you use autoinstalled PROGRAM definitions, you will need to supply a PROGRAM definition for each program that runs in the pipeline. These include the target application program, which normally run under transaction CPIH. The transaction is defined with the attribute TASKDATALOC(ANY). Therefore, when you link-edit the program, you must specify the AMODE(31) option.

A URIMAP resource definition

The URIMAP is used to locate the pipeline that handles Web service requests. Although you can define and install URIMAP resources using RDO, you are advised to create them dynamically.

A WEBSERVICE resource definition

The WEBSERVICE resource defines the execution environment for your application.

Although you can define and install WEBSERVICE resources using RDO, you are advised to create them dynamically, using the Web service binding file that is created by the CICS Web services assistant.

Perform the following steps to create the CICS infrastructure for your service provider:

1. Define the transport infrastructure. Repeat this step for each different transport configuration you need
2. Define the pipeline. Repeat this step for each different pipeline configuration you need.
3. Create a Web service binding file for each application program. Put the file into the pickup directory of the pipeline that you want to use for the service provider.
4. Create a URIMAP and WEBSERVICE resource for each application program. Use the PERFORM PIPELINE SCAN command to do this. Repeat this step whenever you add a Web service binding file to the pickup directory for the PIPELINE.

Your CICS system will now contain the infrastructure needed for each service provider:

- One or more transport infrastructures
- One or more pipelines
- For each Service provider:
 - A URIMAP
 - A WEBSERVICE

You can extend the configuration when you need to do so:

- To define additional transport infrastructure, repeat step 1
- To create additional pipelines, repeat step 2.

- To associate further Web application programs with a pipeline, repeat steps 3 on page 59 through 4 on page 59.

Creating the CICS infrastructure for a service requester

To create the CICS infrastructure for a service requester, you must define and install a number of CICS resources. In many cases, CICS can generate some of these resources automatically.

For a service requester application deployed with the help of the CICS Web services assistant, you will need to define the following:

A PIPELINE resource definition

With its associated *pipeline configuration file*, the PIPELINE resource defines the attributes of the pipeline which is used to process outbound Web service requests, and the responses. Typically, one pipeline can process requests for many different Web services, and when you deploy a new service requester in your CICS system, you will be able to use an existing pipeline.

As well as the configuration file, the PIPELINE resource specifies a *pickup directory*, which contains *Web service binding files*.

When you install a PIPELINE resource, or when you issue a PERFORM PIPELINE SCAN command (using CEMT or the CICS system programming interface), CICS reads the files in the pickup directory, and creates URIMAP and WEBSERVICE resources dynamically.

A WEBSERVICE resource definition

The WEBSERVICE resource defines the execution environment that lets your CICS application program operate as a Web service requester. Although you can define and install WEBSERVICE resources using RDO, you are advised to create them dynamically. There is one WEBSERVICE resource definition for each target Web service.

Perform the following steps to create the CICS infrastructure for your service requester:

1. Define the pipeline. Repeat this step for each different pipeline configuration you need.
2. Create a Web service binding file for each service requester application. Put the file into the pickup directory of the pipeline that you want to use for the service requester.
3. Create a WEBSERVICE resource for each application program. Use the PERFORM PIPELINE SCAN command to do this. Repeat this step whenever you add a Web service binding file to the pickup directory for the PIPELINE.

Your CICS system will now contain the infrastructure needed for each service requester:

- One or more pipelines
- For each Service requester:
 - A WEBSERVICE

You can extend the configuration when you need to do so:

- To create additional pipelines, repeat 1.

- To associate further Web service requester applications with a pipeline, repeat 2 on page 60 through 3 on page 60.

Invoking the Web services assistant using an API

Instead of using JCL, you can write your own Java program to invoke the Web services assistant. A Java API has been provided, along with the JAR files and sample code.

The Java API is described in the Web services assistant: Class Reference Javadoc. It includes comments that explain the classes, and sample code that gives you an example of how to invoke the Web services assistant. The Javadoc also contains a complete list of the JAR files that are required to run your program and where they can be found in HFS.

You can run your Java program on z/OS 1.4 or later. Alternatively you can run the program on the Windows® platform. The following Windows operating system versions are supported:

- Windows 2000 Advanced Server
- Windows 2000 Professional
- Windows 2000 Server
- Windows Server 2003 Enterprise Edition
- Windows Server 2003 Enterprise Edition for 64-bit Extended Systems (32-bit emulation)
- Windows Server 2003 Standard Edition
- Windows Server 2003 Standard Edition for 64-bit Extended Systems (32-bit emulation)
- Windows XP Professional

If you execute the Web services assistant on Windows, you should transfer the generated Web services binding file to HFS in binary mode using FTP or an equivalent process. If you decide to transfer the WSDL file to HFS, use text mode.

You must use Java 1.4 or later when running your program.

Validating SOAP messages

When you use the CICS Web services assistant to deploy your applications, you can specify that the SOAP messages should be validated at run time, to ensure that they conform to the Schema that is contained in the Web service description. You can perform validation in both provider and requester mode.

CICS uses a Java program to validate SOAP messages. Therefore, you must have Java support enabled in your CICS region to perform validation.

If the **LOCALCCSID** system initialization parameter specifies a value other than 037, then the JVM properties file associated with JVMProfile DFHJVMCD must contain the following:

```
com.ibm.cics.soap.validation.local.CCSID=ccsid
```

where *ccsid* is the value of the **LOCALCCSID** system initialization parameter. This value is required, even if you have specified a value for the **CCSID** parameter in the Web services assistant.

Validation of the SOAP message takes place before it is transformed into an application data structure, and when a SOAP message is generated from the application data structure. The SOAP message is validated using the XML schema in the WSDL, before then being validated against the transformation requirements of CICS.

When validation is turned off, CICS does not use the Java program. CICS validates SOAP messages only to the extent that is necessary to confirm that they contain well-formed XML, and to transform them. This means that it is possible for a SOAP message to be successfully validated against the WSDL, but then fail in the runtime environment and vice versa.

Important: During development and testing of your Web service deployment, using full validation will assist in detecting problems in the message exchange between a service requester and a service provider. However, there is a substantial overhead associated with performing complete validation of the SOAP messages, and it is inadvisable to validate messages in a fully tested production application.

To have your SOAP message validated, perform the following steps:

1. Ensure that you have a Web service description associated with your WEBSERVICE resource. This will be the case for WEBSERVICE resource definitions that were created automatically if the Web service description was present in the PIPELINE's pickup directory when the directory was scanned. For WEBSERVICE definitions that were created with RDO, the Web service description is specified with the WSDLFILE attribute.
2. Turn validation on for the WEBSERVICE. Use the following CEMT or SPI command: SET WEBSERVICE(*name*) VALIDATION. For WEBSERVICES that are defined with RDO you can specify whether validation is required or not in the VALIDATION attribute, but you can change this setting after the WEBSERVICE is installed with the SET WEBSERVICE command.

Check the system log to find out if the SOAP message is valid. Message DFHPI1002 indicates that the SOAP message was successfully validated, and message DFHPI1001 indicates that the validation failed.

When you no longer need validation for the Web service, use the following command to turn it off: SET WEBSERVICE(*name*) NOVALIDATION.

DFHLS2WS: high level language to WSDL conversion

The DFHLS2WS procedure generates a Web service description and a Web service binding file from a high-level language data structure. You can use DFHLS2WS when you expose a CICS application program as a service provider.

As per the W3C recommendation for WSDL documents, DFHLS2WS uses a top level wrapper element to contain the body of the SOAP message. The wrapper element takes the name of the WSDL operation and is represented as a `complexType` in the WSDL document.

The job control statements for DFHLS2WS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

Job control statements for DFHLS2WS

JOB Initiates the job.

EXEC Specifies the procedure name (DFHLS2WS).

DFHLS2WS requires sufficient storage to run a Java virtual machine (JVM). You are advised to specify REGION=0M on the EXEC statement.

INPUT.SYSUT1 DD

Specifies the input. The input parameters are usually specified in the input stream. However, they can be defined in a data set, or in a member of a partitioned data set.

Symbolic parameters

The following symbolic parameters are defined in cataloged procedure DFHLS2WS:

JAVADIR=*path*

Specifies the name of the Java directory that is used by DFHLS2WS. The value of this parameter is appended to /usr/lpp/ giving a complete path name of /usr/lpp/*path*.

Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **JAVADIR** parameter.

PATHPREF=*prefix*

Specifies an optional prefix that extends the HFS directory path used on other parameters. The default is the empty string.

Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **JAVADIR** parameter.

SERVICE=*value*

Use this parameter only when directed to do so by IBM support.

TMPDIR=*tmpdir*

Specifies the location of a directory in HFS that DFHLS2WS uses as a temporary work space. The user ID under which the job runs must have read and write permission to this directory.

The default value is /tmp.

TMPFILE=*tmpprefix*

Specifies a prefix that DFHLS2WS uses to construct the names of the temporary workspace files.

The default value is LS2WS

USSDIR=*path*

Specifies the name of the CICS TS directory in the UNIX[®] system services HFS. The value of this parameter is appended to /usr/lpp/cicsts/ giving a complete path name of /usr/lpp/cicsts/*path*

Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **USSDIR** parameter.

The temporary work space

DFHLS2WS creates the following three temporary files during execution:

```
tmpdir/tmpprefix.in  
tmpdir/tmpprefix.out  
tmpdir/tmpprefix.err
```

where

tmpdir is the value specified in the **TMPDIR** parameter
tmpprefix is the value specified in the **TMPFILE** parameter.

The default names for the files (when **TMPDIR** and **TMPFILE** are not specified), are:

```
/tmp/LS2WS.in  
/tmp/LS2WS.out  
/tmp/LS2WS.err
```

Important: DFHLS2WS does not lock access to the generated HFS file names. Therefore, if two or more instances of DFHLS2WS run concurrently, and use the same temporary workspace files, there is nothing to prevent one job overwriting the workspace files while another job is using them. This can lead to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that will avoid this situation. For example, you can use the system symbolic parameter SYSUID to generate workspace file names that are unique to an individual user.

These temporary files are deleted before the end of the job.

Input parameters for DFHLS2WS

If you need any help understanding this syntax diagram, see Syntax notation.



Parameter use

- You can specify the input parameters in any order.
- Each parameter must start on a new line.
- A parameter (and its continuation character, if you use one) must not extend beyond column 72; columns 73 to 80 should contain blanks.
- If a parameter is too long to fit on a single line, use an asterisk (*) character at the end of the line to indicate that the parameter continues on the next line. Everything (including spaces) before the asterisk is considered part of the parameter. For example:

```
WSBIND=wsbinddir*
/app1
```

is equivalent to

```
WSBIND=wsbinddir/app1
```

- A # character in the first character position of the line is a comment character. The line is ignored.

Parameter descriptions

CCSID=value

Specifies the CCSID that is used at run time to encode character data in the

application data structure. The value of this parameter overrides the value of the **LOCALCCSID** system initialization parameter. The *value* must be an EBCDIC CCSID that is supported by Java and z/OS conversion services. If you do not specify this parameter, the application data structure is encoded using the CCSID specified in the system initialization parameter.

You can use this parameter with any mapping level. However, if you want to deploy the generated files, you must apply APAR PK23547 to the CICS region to achieve the minimum runtime level of code to install the Web service binding file.

CHAR-VARYING=NO|NULL

Specifies how character fields in the language structure should be mapped when the mapping level is 1.2. A character field in COBOL is a Picture clause of type X, for example PIC(X) 10; a character field in C/C++ is a character array. This parameter does not apply to Enterprise and Other PL/I language structures. The options you can select are:

NO Character fields are mapped to an `xsd:string` and are processed as fixed length fields. The maximum length of the data is equal to the length of the field.

NULL Character fields are mapped to an `xsd:string` and are processed as null terminated strings. CICS adds a terminating null character when transforming from a SOAP message. The maximum length of the character string is calculated as one character less than the length indicated in the language structure.

CONTID=*value*

In a service provider, specifies the name of the container that holds the top level data structure used to represent a SOAP message.

LANG=COBOL

Specifies that the programming language of the high level language structure is COBOL.

LANG=PLI-ENTERPRISE

Specifies that the programming language of the high level language structure is Enterprise PL/I.

LANG=PLI-OTHER

Specifies that the programming language of the high level language structure is a level of PL/I other than Enterprise PL/I.

LANG=C

Specifies that the programming language of the high level language structure is C.

LANG=CPP

Specifies that the programming language of the high level language structure is C++.

LOGFILE=*value*

The fully qualified HFS name of the file into which DFHLS2WS writes its activity log and trace information. DFHLS2WS creates the file (but not the directory structure) if it does not already exist.

Normally, you will not need to use this file, but it may be requested by the IBM service organization if you encounter problems with DFHLS2WS.

MAPPING-LEVEL={1.0|1.1|1.2}

Specifies the level of mapping that DFHLS2WS should use when generating the

Web service binding file and Web service description. This parameter is available when you apply APAR PK15904. You also need to apply APAR PK23547 if you want to use the 1.2 mapping level option. The options you can select are:

- 1.0 This is the default mapping level.
- 1.1 Use this mapping if you need to regenerate a binding file at this specific level.
- 1.2 At this mapping level you can use the parameter **CHAR-VARYING** to control how character arrays should be processed at run time. **VARYING** and **VARYINGZ** arrays are also supported in PL/I.

For details of what is supported at each level of mapping, see “Mapping levels for the CICS Web services assistant” on page 80.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|CURRENT}

Specifies the minimum CICS runtime environment that the Web service binding file can be deployed into. If you select a level that does not match the other parameters that you have specified, you receive an error message. The options you can select are:

MINIMUM

The lowest possible runtime level of CICS is allocated automatically given the parameters that you have specified.

- 1.0 The generated Web service binding file deploys successfully into a CICS TS 3.1 region that does not have APARs PK15904 and PK23547 applied. You cannot specify the **CHAR-VARYING**, **CCSID**, or **MAPPING-LEVEL** parameters.
- 1.1 The generated Web service binding file deploys successfully into a CICS TS 3.1 region that has at least APAR PK15904 applied. You cannot specify the **CHAR-VARYING** or **CCSID** parameters. You cannot use a mapping level of 1.2 for the **MAPPING-LEVEL** parameter.
- 1.2 The generated Web service binding file deploys successfully into a CICS TS 3.1 region that has both APAR PK15904 and PK23547 applied. You can use any optional parameter at this level.

CURRENT

The generated Web service binding file deploys successfully into a CICS region at the same runtime level as the one you are using to generate the Web service binding file.

PDSLIB=value

Specifies the name of the partitioned data set that contains the high level language data structures to be processed. The data set members used for the request and response are specified in the **REQMEM** and **RESPMEM** parameters respectively.

Restriction: The records in the partitioned data set must have a fixed length of 80 bytes.

PDSCP=value

Specifies the code page used in the partitioned data set members specified in the **REQMEM** and **RESPMEM** parameters, where *value* is a **CCSID** number or a Java code page number. If this parameter is not specified, then the z/OS UNIX System Services code page is used. For example, you could specify **PDSCP=037**.

PGMINT=CHANNEL|COMMAREA

For a service provider, specifies how CICS passes data to the target application program:

CHANNEL

CICS uses a channel interface to pass data to the target application program.

COMMAREA

CICS uses a communication area to pass data to the target application program.

This parameter is ignored when the output from DFHLS2WS is used in a service requester.

PGMNAME=*value*

Specifies the name of the target CICS application program that will be exposed as a Web service. This is the program that the CICS Web service support will link to.

REQMEM=*value*

Specifies the name of the partitioned data set member which contains the high level language structure for the Web service request:

- For a service provider, the Web service request is the input to the application program
- For a service requester, the Web service request is the output from the application program

REQUEST-NAMESPACE=*value*

Specifies the namespace of the XML schema for the request message in the generated Web service description. If you do not specify this parameter, CICS generates a namespace automatically.

RESPMEM=*value*

Specifies the name of the partitioned data set member which contains the high level language structure for the Web service response:

- For a service provider, the Web service response is the output from the application program
- For a service requester, the Web service response is the input to the application program

If there is no response (that is, for one way messages) omit this parameter.

RESPONSE-NAMESPACE=*value*

Specifies the namespace of the XML schema for the response message in the generated Web service description. If you do not specify this parameter, CICS generates a namespace automatically.

STRUCTURE=(*request,response*)

For C and C++ only, specifies the names of the high level structures contained in the partitioned data set members specified in the REQMEM and RESPMEM parameters:

request

specifies the name of the high level structure containing the request when the REQMEM parameter is specified. The default value is DFHREQUEST.

The partitioned data set member must contain a high level structure with the name that you specify (or a structure named DFHREQUEST if you do not specify a name).

response

specifies the name of the high level structure containing the response when the RESPMEM parameter is specified. The default value is DFHRESPONSE.

If you specify a value, the partitioned data set member must contain a high level structure with the name that you specify (or a structure named DFHRESPONSE if you do not specify a name).

SYNCONRETURN=NO|YES

specifies whether the remote Web service can issue a syncpoint.

NO The remote Web service cannot issue a syncpoint. This value is the default. If the remote Web service issues a syncpoint, it fails with an ADPL abend.

YES The remote Web service can issue a syncpoint. If you select YES, the remote task is committed as a separate unit of work when control returns from the remote Web service. If the remote Web service updates a recoverable resource and a failure occurs after it returns, the update to that resource cannot be backed out.

TRANSACTION=name

In a service provider, this parameter specifies the 1-4 character name of an alias transaction that can start the pipeline. The value of this parameter is used to define the TRANSACTION attribute of the URIMAP resource when it is created automatically using the PIPELINE scan command.

Acceptable characters:

A-Z a-z 0-9 \$ @ # _ < >

URI=value

In a service provider, this parameter specifies the relative URI that a client will use to access the Web service. CICS uses the value specified when it generates a URIMAP resource from the Web service binding file created by DFHLS2WS: the parameter specifies the path component of the URI to which the URIMAP definition applies.

USERID=id

In a service provider, this parameter specifies a 1-8 character user ID which can be used by any Web client. For an application-generated response or a Web service, the alias transaction is attached under this user ID. The value of this parameter is used to define the USERID attribute of the URIMAP resource when it is created automatically using the PIPELINE scan command.

Acceptable characters:

A-Z a-z 0-9 \$ @ #

WSBIND=value

The fully qualified HFS name of the Web service binding file. DFHLS2WS creates the file (but not the directory structure) if it does not already exist.

WSDL=value

The fully qualified HFS name of the file into which the Web service description is written. DFHLS2WS creates the file (but not the directory structure) if it does not already exist.

WSDLCP=LOCAL|UTF-8

Specifies the code page that is used to generate the WSDL document.

LOCAL

This value specifies that the WSDL document is generated using the local code page and no encoding tag is generated in the WSDL document.

UTF-8 This value specifies that the WSDL document is generated using the UTF-8 code page. An encoding tag is generated in the WSDL document. If you specify this option, you must ensure that the encoding remains correct when copying the WSDL document between different platforms.

Other information

- The user ID under which DFHLS2WS runs must be defined to OMVS. The user ID must have read permission to the CICS HFS file structure and PDS libraries, and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **WSDL** parameters.
- The user ID must have a sufficiently large storage allocation to run Java.

Example

```
//LS2WS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHLS2WS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
LOGFILE=/u/exampleapp/wsbinding/inquireSingle.log
MAPPING-LEVEL=1.0
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
WSBIND=/u/exampleapp/wsbinding/inquireSingle.wsbinding
WSDL=/u/exampleapp/wsd1/inquireSingle.wsd1
WSDLCP=LOCAL
/*
```

DFHWS2LS: WSDL to high level language conversion

Cataloged procedure DFHWS2LS generates a high level language data structure and a Web service binding file from a Web service description. You can use DFHWS2LS when you expose a CICS application program as a service provider or when you construct a service requester.

Job control statements for DFHWS2LS

JOB Initiates the job.

EXEC Specifies the procedure name (DFHWS2LS).

DFHWS2LS requires sufficient storage to run a Java virtual machine (JVM). You are advised to specify **REGION=0M** on the EXEC statement.

INPUT.SYSUT1 DD

Specifies the input. The input parameters are usually specified in the input stream. However, they can be defined in a data set, or in a member of a partitioned data set.

Symbolic parameters

The following symbolic parameters are defined in cataloged procedure DFHWS2LS:

JAVADIR=*path*

Specifies the name of the Java directory that is used by DFHWS2LS. The value of this parameter is appended to `/usr/lpp/` giving a complete path name of `/usr/lpp/path`.

Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **JAVADIR** parameter.

PATHPREF=*prefix*

Specifies an optional prefix that extends the HFS directory path used on other parameters. The default is the empty string.

Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **JAVADIR** parameter.

TMPDIR=*tmpdir*

Specifies the location of a directory in HFS that DFHWS2LS uses as a temporary work space. The user ID under which the job runs must have read and write permission to this directory.

The default value is `/tmp`.

TMPFILE=*tmpprefix*

Specifies a prefix that DFHWS2LS uses to construct the names of the temporary workspace files.

The default value is `WS2LS`.

USSDIR=*path*

Specifies the name of the CICS TS directory in the UNIX system services HFS. The value of this parameter is appended to `/usr/lpp/cicsts/` giving a complete path name of `/usr/lpp/cicsts/path`.

Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **USSDIR** parameter.

SERVICE=*value*

Use this parameter only when directed to do so by IBM support.

The temporary work space

DFHWS2LS creates the following three temporary files during execution:

```
tmpdir/tmpprefix.in  
tmpdir/tmpprefix.out  
tmpdir/tmpprefix.err
```

where

tmpdir is the value specified in the **TMPDIR** parameter

tmpprefix is the value specified in the **TMPFILE** parameter.

The default names for the files (when **TMPDIR** and **TMPFILE** are not specified), are:

```
/tmp/WS2LS.in  
/tmp/WS2LS.out
```

/tmp/WS2LS.err

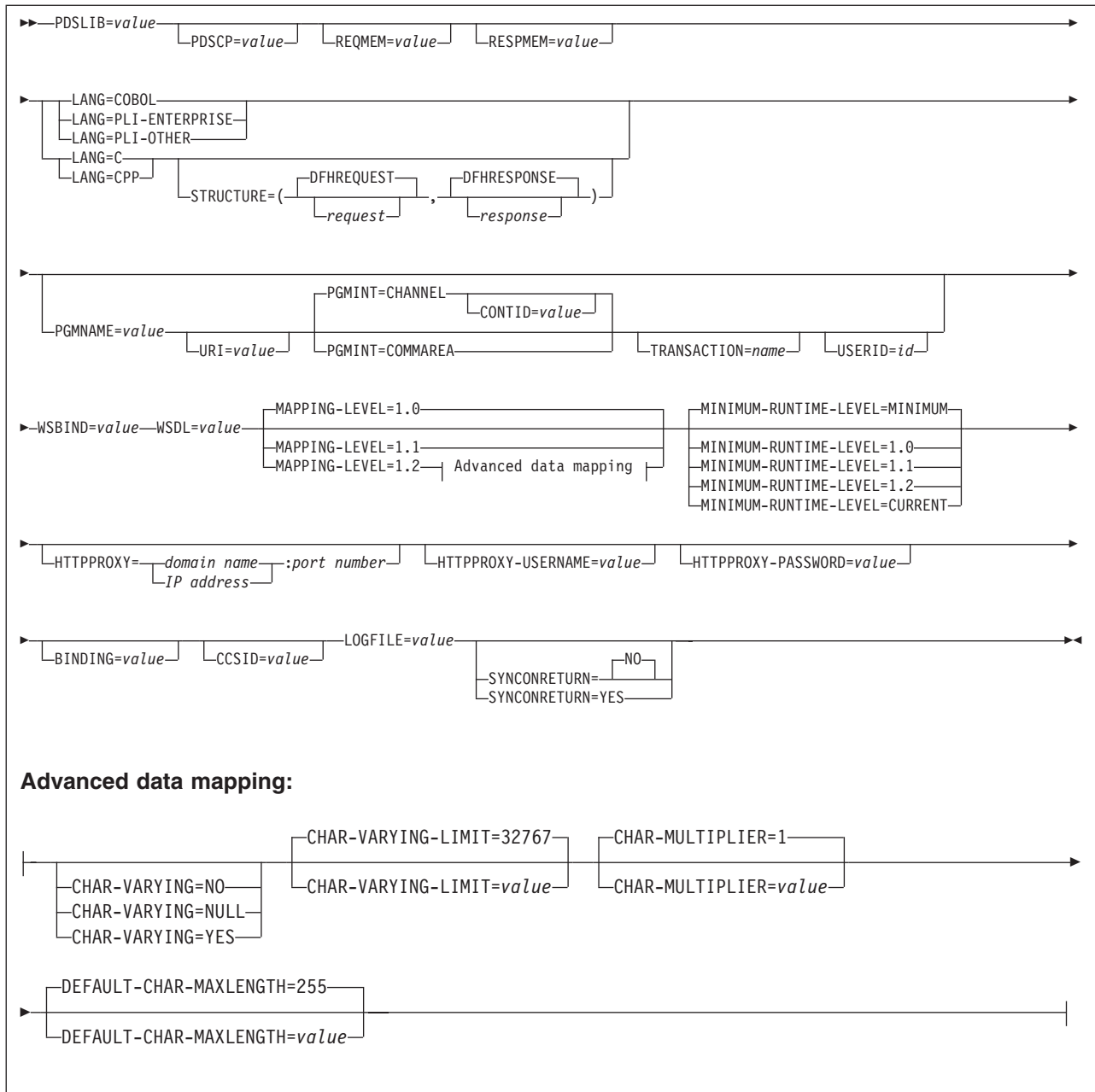
Important: DFHWS2LS does not lock access to the generated HFS file names. Therefore, if two or more instances of DFHWS2LS run concurrently, and use the same temporary workspace files, there is nothing to prevent one job overwriting the workspace files while another job is using them. This can lead to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that will avoid this situation. For example, you can use the system symbolic parameter SYSUID to generate workspace file names that are unique to an individual user.

These temporary files are deleted before the end of the job.

Input parameters for DFHWS2LS

If you need any help understanding this syntax diagram, see Syntax notation.



Parameter use

- You can specify the input parameters in any order.
- Each parameter must start on a new line.
- A parameter (and its continuation character, if you use one) must not extend beyond column 72; columns 73 to 80 should contain blanks.
- If a parameter is too long to fit on a single line, use an asterisk (*) character at the end of the line to indicate that the parameter continues on the next line. Everything (including spaces) before the asterisk is considered part of the parameter. For example:

```
WSBIND=wsbinddir*
/app1
```

is equivalent to

WSBIND=wsbinddir/app1

- A # character in the first character position of the line is a comment character. The line is ignored.

Parameter descriptions

BINDING=*value*

If the Web service description contains more than one <binding> element, use this parameter to specify which one is to be used to generate the language structure and Web service binding file. Specify the value of the name attribute that is used on the <binding> element in the Web service description.

CCSID=*value*

Specifies the CCSID that is used at run time to encode character data in the application data structure. The value of this parameter overrides the value of the **LOCALCCSID** system initialization parameter. The *value* must be an EBCDIC CCSID that is supported by Java and z/OS conversion services. If you do not specify this parameter, the application data structure is encoded using the CCSID specified in the system initialization parameter.

You can use this parameter with any mapping level. However, if you want to deploy the generated files, you must apply APAR PK23547 to the CICS region to achieve the minimum runtime level of code to install the Web service binding file.

CHAR-MULTIPLIER=1|*value*

Specifies the number of bytes to allow for each character when the mapping level is 1.2. The *value* of this parameter can be a positive integer in the range of 1 to 2147483647. All nonnumeric character-based mappings, are subject to this multiplier. Binary, numeric, zoned and packed decimal fields are not subject to this multiplier.

This parameter can be useful if, for example, you are planning to use DBCS characters where you could opt for a multiplier of 3 to allow space for potential shift-out and shift-in characters around every double byte character at run time.

CHAR-VARYING=NO|NULL|YES

Specifies how variable length character data is mapped. Variable length character data is where the minimum and maximum length of a field is different. This parameter can only be used when the mapping level is 1.2. If you do not specify this parameter, the default mapping depends on the language that is specified. These defaults are described in the mappings for each language and XML schema in “High level language and XML schema mapping” on page 83. The options that you can select are:

- NO** Variable length character data is mapped as fixed length strings.
- NULL** Variable length character data is mapped to null terminated strings.
- YES** Variable length character data is mapped to a CHAR VARYING data type in PL/I. In the COBOL, C and C++ languages, variable length character data is mapped to an equivalent representation that comprises of two related elements - data length and the data.

CHAR-VARYING-LIMIT=32767|*value*

Specifies the maximum size of variable length character data that is mapped to the language structure. If the character data is larger than the value specified in this parameter, it is mapped to a container and the container name is used in the generated language structure. The *value* can range from 0 to the default 32767 bytes.

This parameter can only be used when the mapping level is 1.2.

CONTID=*value*

In a service provider, specifies the name of the container that holds the top level data structure used to represent a SOAP message.

DEFAULT-CHAR-MAXLENGTH=255|*value*

Specifies the default field length of character data in characters for mappings where no length is implied in the Web service description document. The *value* of this parameter can be a positive integer in the range of 1 to 2147483647.

You can only use this parameter when the mapping level is 1.2.

HTTPPROXY={*domain name|IP address***};***port number*

If your WSDL contains references to other WSDL files that are located on the internet, and the system on which you are running DFHWS2LS uses a proxy server to access the internet, specify the domain name or IP address, and port number, of the proxy server. For example:

```
HTTPPROXY=proxy.example.com:8080
```

In other cases, this parameter is not required.

HTTPPROXY-USERNAME=*value*

Specifies the HTTP proxy username that should be used in conjunction with **HTTPPROXY-PASSWORD** if the system on which you are running DFHWS2LS uses a HTTP proxy server to access the Internet, and the HTTP proxy server uses basic authentication. You can only use this parameter when you also specify **HTTPPROXY**.

HTTPPROXY-PASSWORD=*value*

Specifies the HTTP proxy password that should be used in conjunction with **HTTPPROXY-USERNAME** if the system on which you are running DFHWS2LS uses a HTTP proxy server to access the Internet, and the HTTP proxy server uses basic authentication. You can only use this parameter when you also specify **HTTPPROXY**.

LANG=COBOL

Specifies that the programming language of the high level language structure is COBOL.

LANG=PLI-ENTERPRISE

Specifies that the programming language of the high level language structure is Enterprise PL/I.

LANG=PLI-OTHER

Specifies that the programming language of the high level language structure is a level of PL/I other than Enterprise PL/I.

LANG=C

Specifies that the programming language of the high level language structure is C.

LANG=CPP

Specifies that the programming language of the high level language structure is C++.

LOGFILE=*value*

The fully qualified HFS name of the file into which DFHWS2LS writes its activity log and trace information. DFHWS2LS creates the file (but not the directory structure) if it does not already exist.

Normally you will not need to use this file, but it may be requested by the IBM service organization if you encounter problems with DFHWS2LS.

MAPPING-LEVEL={1.0|1.1|1.2}

Specifies the level of mapping that DFHWS2LS should use when generating the Web service binding file and language structure. This parameter is available when you apply APAR PK15904. You also need to apply APAR PK23547 if you want to use the 1.2 mapping level option. The options you can select are:

- 1.0** This is the default mapping level.
- 1.1** XML attributes, <list> data types, and <union> data types are mapped to the language structure. Character and binary data that has a maximum length of more than 32,767 bytes is mapped to a container. The container name is created in the language structure.
- 1.2** Use the parameters **CHAR-VARYING** and **CHAR-VARYING-LIMIT** to control how character data is mapped and processed at run time. If you do not specify either of these parameters, binary and character data that has a maximum length less than 32768 bytes is mapped to a VARYING structure for all languages except C++, where character data is mapped to a null terminated string.

For details of what is supported at each level of mapping, see “Mapping levels for the CICS Web services assistant” on page 80.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|CURRENT}

Specifies the minimum CICS runtime environment that the Web service binding file can be deployed into. If you select a level that does not match the other parameters that you have specified, you receive an error message. The options you can select are:

MINIMUM

The lowest possible runtime level of CICS is allocated automatically given the parameters that you have specified.

- 1.0** The generated Web service binding file deploys successfully into a CICS TS 3.1 region that does not have APARs PK15904 and PK23547 applied. You cannot specify the **CCSID** or **MAPPING-LEVEL** parameter, or any other optional parameters that rely on the **MAPPING-LEVEL** parameter.
- 1.1** The generated Web service binding file deploys successfully into a CICS TS 3.1 region that has at least APAR PK15904 applied. You cannot specify the **CCSID** parameter or use a mapping level of 1.2 for the **MAPPING-LEVEL** parameter. You cannot specify any optional parameters that rely on the 1.2 level of mapping.
- 1.2** The generated Web service binding file deploys successfully into a CICS TS 3.1 region that has both APAR PK15904 and PK23547 applied. You can use any optional parameter at this level.

CURRENT

The generated Web service binding file deploys successfully into a CICS region at the same runtime level as the one you are using to generate the Web service binding file.

PDSLIB=*value*

Specifies the name of the partitioned data set that contains the generated high level language. The data set members used for the request and response are specified in the **REQMEM** and **RESPMEM** parameters respectively.

PDSCP=*value*

Specifies the code page used in the partitioned data set members specified in the REQMEM and RESPMEM parameters, where *value* is a CCSID number or a Java code page number. If this parameter is not specified, then the z/OS UNIX System Services code page is used. For example, you could specify PDSCP=037.

PGMINT=CHANNEL|COMMAREA

For a service provider, specifies how CICS passes data to the target application program:

CHANNEL

CICS uses a channel interface to pass data to the target application program.

COMMAREA

CICS uses a communication area to pass data to the target application program.

This parameter is ignored when the output from DFHWS2LS is used in a service requester.

PGMNAME=*value*

This parameter specifies the name of a CICS program.

When DFHWS2LS is being used to generate a Web service binding file that will be used in a service provider, this parameter must be supplied. It specifies the name of the application program that is being exposed as a Web service.

When DFHWS2LS is being used to generate a Web service binding file that will be used in a service requester, this parameter must be omitted.

REQMEM=*value*

Specifies a 1 - 6 character prefix that DFHWS2LS uses to generate the names of the partitioned data set members that will contain the high level language structures for the Web service request:

- For a service provider, the Web service request is the input to the application program
- For a service requester, the Web service request is the output from the application program

DFHWS2LS generates a partitioned data set member for each operation. It generates the member name by appending a two digit number to the prefix.

Although this parameter is optional, you must specify it if the Web service description contains a definition of a request.

RESPMEM=*value*

Specifies a 1 - 6 character prefix that DFHWS2LS uses to generate the names of the partitioned data set members that will contain the high level language structures for the Web service response:

- For a service provider, the Web service response is the output from the application program
- For a service requester, the Web service response is the input to the application program

DFHWS2LS generates a partitioned data set member for each operation. It generates the member name by appending a two digit number to the prefix.

If there is no response (that is, for one way messages) omit this parameter.

STRUCTURE=(*request,response*)

For C and C++ only, specifies how the names of the request and response structures are generated.

The generated request and response structures are given names of *requestnn* and *responsenn* where *nn* is a numeric suffix that is generated to distinguish the structures for each operation.

If one or both names is omitted, the structures have the same name as the partitioned data set member names generated from the REQMEM and RESPMEM parameters that you specify.

SYNCONRETURN=**NO|YES**

specifies whether the remote Web service can issue a syncpoint.

NO The remote Web service cannot issue a syncpoint. This value is the default. If the remote Web service issues a syncpoint, it fails with an ADPL abend.

YES The remote Web service can issue a syncpoint. If you select YES, the remote task is committed as a separate unit of work when control returns from the remote Web service. If the remote Web service updates a recoverable resource and a failure occurs after it returns, the update to that resource cannot be backed out.

TRANSACTION=*name*

In a service provider, this parameter specifies the 1-4 character name of an alias transaction that can start the pipeline. The value of this parameter is used to define the TRANSACTION attribute of the URIMAP resource when it is created automatically using the PIPELINE scan command.

Acceptable characters:

A-Z a-z 0-9 \$ @ # _ < >

URI=*value*

In a service provider, this parameter specifies the relative URI that a client will use to access the Web service. CICS uses the value specified when it generates a URIMAP resource from the Web service binding file created by DFHWS2LS: the parameter specifies the path component of the URI to which the URIMAP definition applies.

In a service requester, the URI of the target Web service is **not** specified with this parameter: the URI specified in the Web service description is used, although you can override that with the URI option on the EXEC CICS INVOKE WEBSERVICE command.

USERID=*id*

In a service provider, this parameter specifies a 1-8 character user ID which can be used by any Web client. For an application-generated response or a Web service, the alias transaction is attached under this user ID. The value of this parameter is used to define the USERID attribute of the URIMAP resource when it is created automatically using the PIPELINE scan command.

Acceptable characters:

A-Z a-z 0-9 \$ @ #

WSBIND=*value*

The fully qualified HFS name of the Web service binding file. DFHWS2LS creates the file (but not the directory structure) if it does not already exist.

WSDL=value

The fully qualified HFS name of the file that contains the Web service description.

Other information

- The user ID under which DFHWS2LS runs must be defined to OMVS. The user ID must have read permission to the CICS HFS file structure and PDS libraries, and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **WSDL** parameters.
- The user ID must have a sufficiently large storage allocation to run Java.

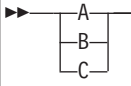
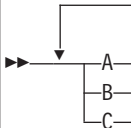
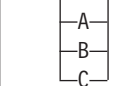
Example

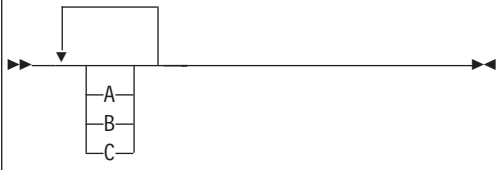

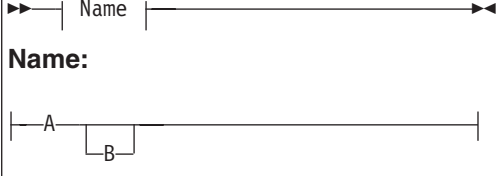

```
//WS2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHWS2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/inquireSingle.log
MAPPING-LEVEL=1.0
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsd1/inquireSingle.wsd1
/*
```

Syntax notation

Syntax notation specifies the permissible combinations of options or attributes that you can specify on CICS commands, resource definitions, and many other things.

The conventions used in the syntax notation are:

Notation	Explanation
	Denotes a set of required alternatives. You must specify one (and only one) of the values shown.
	Denotes a set of required alternatives. You must specify at least one of the values shown. You can specify more than one of them, in any sequence.
	Denotes a set of optional alternatives. You can specify none, or one, of the values shown.

Notation	Explanation
	Denotes a set of optional alternatives. You can specify none, one, or more than one of the values shown, in any sequence.
	Denotes a set of optional alternatives. You can specify none, or one, of the values shown. A is the default value that is used if you do not specify anything.
	A reference to a named section of syntax notation.
	A= denote characters that should be entered exactly as shown. <i>value</i> denotes a variable, for which you should specify an appropriate value.

Mapping levels for the CICS Web services assistant

A mapping is the set of rules used to determine how information is converted between language structures and the Web service description (WSDL) document.

When you run the Web services assistant jobs DFHWS2LS and DFHLS2WS, you can use the **MAPPING-LEVEL** parameter to set a level of mapping that can map language structures and elements in WSDL documents with increasing levels of sophistication.

Mapping levels 1.0 and 1.1 are available when you apply APAR PK15904. Mapping level 1.2 is available when you apply APAR PK23547. Each level of mapping inherits the functionality of the previous mapping, with the highest level of mapping offering the best capabilities available. This includes providing you with more control over how data should be converted at run time, as well as lifting restrictions on support for certain data types and XML elements. These restrictions are explained in “High level language and XML schema mapping” on page 83 for each supported high level language.

Mapping level 1.0

This is the default mapping level. This ensures that you can regenerate your older Web service binding files with a newer level of the Web services assistant without having to make application changes. It is provided for backwards compatibility and should not be used for new applications.

In the default mapping:

- DFHLS2WS interprets character and binary fields in the language structure as fixed length fields and maps the fields to XML elements that have a `maxLength` attribute. At run time the fields in the language structure are filled with spaces or nulls if there is insufficient data. In provider mode, if there is too much data for the field, CICS generates a SOAP fault. In requester mode, CICS returns a RESP2 code of 14 on the INVOKE WEBSERVICE command.
- DFHWS2LS maps character and binary data types in the XML schema to fixed length fields in the language structure. For example, the following partial XML schema:

```
<xsd:element name="example">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="33000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

would appear in a COBOL language structure as:

```
15 example    PIC X(33000)
```

- CICS encodes and decodes data in the `hexBinary` format but not in `base64Binary` format. DFHWS2LS maps `Base64Binary` data to a fixed length character field, the contents of which must be encoded or decoded by the application program.
- DFHWS2LS ignores XML attributes during processing.

Mapping level 1.1

At this level of mapping there are improvements to DFHWS2LS when mapping XML character and binary data types, in particular when mapping data of variable length. This is where `maxLength` and `minLength` attributes are defined with different values in the XML schema. Data is handled in the following ways:

- Variable length binary data types map to a container. A 16-byte field is created in the language structure to store the name of the container. At run time, the binary data is stored in a container and the container name is put in the language structure.
- Variable length character data types that have a maximum length that is greater than 32,767 bytes map to a container. A 16-byte field is created in the language structure to store the name of the container. At run time, the character data is stored in a container and the container name is put in the language structure.
- Character and binary data types that have a fixed length that is greater than 16MB map to a container for all languages except PL/I. In PL/I, fixed length character and binary data types that are greater than 32,767 bytes are mapped to a container. A 16-byte field is created in the language structure to store the name of the container. At run time, the fixed length data is stored in a container and the container name is put in the language structure.

As containers are variable in length, fixed length data that is mapped to a container is not padded with spaces or nulls, or truncated, to match the fixed length specified in the Web service description. If the length of the data is important, you can either write your application to check it or turn SOAP validation on in the CICS region. Note that there is a significant performance impact when using SOAP validation.

- Character and binary data types that have a fixed length of less than 16MB map to fixed length fields for all languages except PL/I. In PL/I, fixed length character and binary data types that are 32,767 bytes or less map to fixed length fields.
- XML schema <list> and <union> data types map to character fields.
- Base64Binary data types in the XML schema map to a field in the language structure. The size of the field is calculated using the formula: $4 \times (\text{ceil}(z/3))$ where z is the length of the data type in the XML schema and $\text{ceil}(x)$ is the smallest integer greater than or equal to x . If the length of z is greater than 24566 bytes, the resulting language structure would fail to compile. If you have base64Binary data that is greater than 24566 bytes, it is recommended that you use a mapping level of 1.2. This allows you to map the base64Binary data to a container instead of using a field in the language structure.
- Schema-defined XML attributes are mapped rather than ignored. A maximum of 255 attributes are allowed for each XML element. See “Support for XML attributes” on page 107 for further information.
- The `xsi:nil` attribute is supported. See “Support for XML attributes” on page 107 for further information.

Mapping level 1.2

At this level of mapping you can use additional parameters in DFHWS2LS and DFHLS2WS to control how character and binary data is transformed at run time. If you decide to use the **CHAR-MULTIPLIER** parameter in DFHWS2LS, be aware that the rules below apply after the value of this parameter is used to calculate the amount of space required for character data.

- DFHWS2LS maps variable length character data types that have a maximum length of more than 32,767 bytes to a container. You can use the **CHAR-VARYING-LIMIT** parameter to set a lower limit. A 16-byte field is created in the language structure to store the name of the container. At run time, the character data is stored in a container and the container name is put in the language structure.
- DFHWS2LS maps variable length character data types that have a maximum length of less than 32,768 bytes to a VARYING structure for all languages except C/C++ and Enterprise PL/I. In C/C++ these data types are mapped to null terminated strings, and in Enterprise PL/I these data types are mapped to VARYINGZ structures. You can use the **CHAR-VARYING** parameter to select how variable length character data is mapped.
- DFHWS2LS maps variable length binary data that has a maximum length of less than 32,768 bytes to a VARYING structure for all languages. If the maximum length is equal to or greater than 32,768 bytes the data is mapped to a container. A 16-byte field is created in the language structure to store the name of the container. At run time, the binary data is stored in a container and the container name is put in the language structure.
- In DFHLS2WS, character fields map to an `xsd:string` data type and can be processed as fixed length fields or null terminated strings at run time. You can use the **CHAR-VARYING** parameter to select how variable length character data should be handled at run time for all languages except PL/I.
- CICS encodes and decodes base64Binary data as well as hexBinary data. DFHWS2LS maps base64Binary data types to a container if the maximum length of the data is greater than 32,767 bytes or when the length is not defined. If the length of the data is 32,767 or less, the base64Binary data type is mapped to a VARYING structure for all languages.

If you have character data types in the XML schema that do not have a length associated with them, you can assign a default length using the **DEFAULT-CHAR-MAXLENGTH** parameter in DFHWS2LS.

High level language and XML schema mapping

Web service descriptions use XML Schema to describe the use of simple and complex data types within a SOAP message. When utility programs DFHLS2WS and DFHWS2LS generate Web services description from high level language data structures, and high level language data structures from Web services descriptions, they generate a mapping between the data types used in the two places.

- Program DFHLS2WS maps high level language data types to XML Schema `<simpleType>` elements.
- Program DFHWS2LS maps `<simpleType>` elements to high level language data types.

The two mapping are not symmetrical. This means:

- If you process a language data structure with DFHLS2WS, and then process the resulting Web service description with DFHWS2LS, you should not expect the final data structure to be the same as the one you started with. However, the final data structure is logically equivalent to the one that you submitted to DFHLS2WS.
- If you process a Web service description with DFHWS2LS, and then process the resulting language data structure with DFHLS2WS, you should not expect the XML Schema in the final Web service description to be the same as the one you started with.
- In some cases, DFHWS2LS generates language data types that are not supported by DFHLS2WS.

Language structures processed by DFHLS2WS must be correctly coded according to the rules of the language as implemented in the language compilers that CICS supports.

DFHWS2LS supports Web services descriptions that conform to WSDL version 1.1, with the following restrictions:

- Only SOAP bindings that use literal encodings are supported. This means that the use attribute must be set to a value of `literal`. `use="encoded"` is not supported.
- The only transport protocols supported by DFHWS2LS are HTTP, HTTPS and WebSphere MQ Series.
- Data type definitions must be encoded using the XML Schema Definition language (XSD). Within the schema, data types used in the SOAP message must be explicitly declared. DFHWS2LS does not support data types in the SOAP message that are derived from other data types in the schema and that are not declared.

DFHWS2LS does not support:

- the `<any>` element
- the `maxOccurs` and `minOccurs` attributes on the `<sequence>`, `<all>` and `<choice>` elements.
- abstract types (except as nonterminal types in an inheritance hierarchy)
- the `anyType` type

- cyclic references (for example, where type A contains type B which, in turn, contains type A)

When the mapping level is 1.1 or higher, DFHWS2LS supports:

- the <list> and <union> elements
- the anySimpleType type

DFHWS2LS can process Web service descriptions that contain the <attribute> element, but the element is ignored unless the mapping level is 1.1. For information on what data types are supported for each mapping level, see “Mapping levels for the CICS Web services assistant” on page 80.

- The length of some keywords within the Web services description is limited. For example, operation, binding and part names are limited to 255 characters in length (in some cases the maximum operation name length may be slightly shorter).
- Only one service element is supported for each binding element.
- Any SOAP faults defined in the Web service description are ignored. If you want a service provider application to send a SOAP fault message, use the EXEC CICS SOAPFAULT command.

Characters such as the opening angle bracket (<) are reserved in XML. CICS handles these characters correctly when it maps application data to the elements within a SOAP body. For example, < is mapped to <;.

The null character (X'00') is not permitted in XML. If CICS maps application data containing this character into a SOAP body, it is treated as a null-terminated string.

COBOL and XML Schema mapping

Utility programs DFHLS2WS and DFHWS2LS support mappings between COBOL data structures and the XML Schema definitions that are included in each Web service description.

COBOL to XML Schema

COBOL names are converted to XML names according to the following rules:

1. Duplicate names are made unique by the addition of one or more numeric digits.
For example, two instances of year become year and year1.
2. Hyphens are replaced by underscore characters. Strings of contiguous hyphens are replaced by contiguous underscores.
For example, current-user--id becomes current_user__id.
3. Segments of names that are delimited by hyphens and that contain only upper case characters are converted to lower case.
For example, CA-REQUEST-ID becomes ca_request_id.
4. A leading underscore character is added to names that start with a numeric character.
For example, 9A-REQUEST-ID becomes _9a_request_id.

DFHLS2WS maps COBOL data description elements to schema elements according to the following table. COBOL data description elements that are not shown in the table are not supported by DFHLS2WS. The following restrictions also apply:

- Data description items with level-numbers of 66 and 77 are not supported. Data description items with a level-number of 88 are ignored.
- The following clauses on data description entries are not supported:
 - OCCURS DEPENDING ON
 - OCCURS INDEXED BY
 - REDEFINES
 - RENAMES (that is level 66)
 - DATE FORMAT
- The following clauses on data description items are ignored:
 - BLANK WHEN ZERO
 - JUSTIFIED
 - VALUE
- The SIGN clause SIGN TRAILING is supported. The SIGN clause SIGN LEADING is only supported when the mapping level specified in DFHLS2WS is 1.2.
- SEPARATE CHARACTER is supported at a mapping level of 1.2 for both SIGN TRAILING and SIGN LEADING clauses.
- The following phrases on the USAGE clause are not supported:
 - OBJECT REFERENCE
 - POINTER
 - FUNCTION-POINTER
 - PROCEDURE-POINTER
- The following phrases on the USAGE clause are supported at a mapping level of 1.2.
 - COMPUTATIONAL-1
 - COMPUTATIONAL-2
- The only PICTURE characters supported for DISPLAY and COMPUTATIONAL-5 data description items are 9 and S.
- The PICTURE characters supported for PACKED-DECIMAL data description items are 9, S, and V.
- If the **MAPPING-LEVEL** parameter is set to 1.2 and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an xsd:string and are processed as null terminated strings.

COBOL data description	Schema simpleType
PIC X(<i>n</i>) PIC A(<i>n</i>) PIC G(<i>n</i>) DISPLAY-1 PIC N(<i>n</i>)	<pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="<i>n</i>"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY	<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> <xsd:minInclusive value="-<i>n</i>"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>

COBOL data description	Schema simpleType
PIC S9(z) DISPLAY where $5 \leq z \leq 9$	<pre data-bbox="594 222 1003 380"><xsd:simpleType> <xsd:restriction base="xsd:int"> <xsd:minInclusive value="-n"/> <xsd:maxInclusive value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p data-bbox="594 415 1401 468">where n is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC S9(z) DISPLAY where $9 < z$	<pre data-bbox="594 489 1013 646"><xsd:simpleType> <xsd:restriction base="xsd:long"> <xsd:minInclusive value="-n"/> <xsd:maxInclusive value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p data-bbox="594 682 1401 735">where n is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<pre data-bbox="594 756 1122 913"><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p data-bbox="594 949 1401 1001">where n is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9(z) DISPLAY where $5 \leq z \leq 9$	<pre data-bbox="594 1022 1096 1180"><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p data-bbox="594 1215 1401 1268">where n is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9(z) DISPLAY where $9 < z$	<pre data-bbox="594 1289 1109 1446"><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p data-bbox="594 1482 1401 1535">where n is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC S9(n) COMP PIC S9(n) COMP-4 PIC S9(n) COMP-5 PIC S9(n) BINARY where $n \leq 4$.	<pre data-bbox="594 1556 1024 1650"><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>
PIC S9(n) COMP PIC S9(n) COMP-4 PIC S9(n) COMP-5 PIC S9(n) BINARY where $5 \leq n \leq 9$.	<pre data-bbox="594 1738 1003 1833"><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>

COBOL data description	Schema simpleType
PIC S9(<i>n</i>) COMP PIC S9(<i>n</i>) COMP-4 PIC S9(<i>n</i>) COMP-5 PIC S9(<i>n</i>) BINARY where $9 < n$.	<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></pre>
PIC 9(<i>n</i>) COMP PIC 9(<i>n</i>) COMP-4 PIC 9(<i>n</i>) COMP-5 PIC 9(<i>n</i>) BINARY where $n \leq 4$.	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>
PIC 9(<i>n</i>) COMP PIC 9(<i>n</i>) COMP-4 PIC 9(<i>n</i>) COMP-5 PIC 9(<i>n</i>) BINARY where $5 \leq n \leq 9$.	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></pre>
PIC 9(<i>n</i>) COMP PIC 9(<i>n</i>) COMP-4 PIC 9(<i>n</i>) COMP-5 PIC 9(<i>n</i>) BINARY where $9 < n$.	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></pre>
PIC S9(<i>m</i>)V9(<i>n</i>) COMP-3	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="p"/> <xsd:fractionDigits value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p>where $p = m + n$.</p>
PIC 9(<i>m</i>)V9(<i>n</i>) COMP-3	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="p"/> <xsd:fractionDigits value="n"/> <xsd:minInclusive value="0"/> </xsd:restriction> </xsd:simpleType></pre> <p>where $p = m + n$.</p>
PIC S9(<i>m</i>)V9(<i>n</i>) DISPLAY Supported at mapping level 1.2 only	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="p"/> <xsd:fractionDigits value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p>where $p = m + n$.</p>
COMP-1 Supported at mapping level 1.2 only	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>
COMP-2 Supported at mapping level 1.2 only	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>

XML schema to COBOL

The CICS Web services assistant generates unique and valid names for COBOL variables from the schema element names using the following rules:

1. COBOL reserved words are prefixed with 'X'.
For example, DISPLAY becomes XDISPLAY.
2. Characters other than A-Z, a-z, 0-9 or hyphen are replaced with 'X'.
For example, monthly_total becomes monthlyXtotal.
3. If the last character is a hyphen, it is replaced with 'X'.
For example, ca-request- becomes ca-requestX.
4. If the schema specifies that the variable has varying cardinality (that is, minOccurs and maxOccurs are specified with different values), and the schema element name is longer than 23 characters, it is truncated to that length.
If the schema specifies that the variable has fixed cardinality, and the schema element name is longer than 28 characters, it is truncated to that length.
5. Duplicate names in the same scope are made unique by the addition of one or two numeric digits to the second and subsequent instances of the name.
For example, three instances of year become year, year1 and year2.
6. Five characters are reserved for the strings -cont or -num which are used when the schema specifies that the variable has varying cardinality; that is, when minOccurs and maxOccurs are specified.
For more information, see “Variable arrays of elements” on page 104.
7. For attributes, the previous rules are applied to the attribute name. The prefix attr- is added to the attribute name, and this is followed by -value or -exist. If the total length is longer than 28 characters, the attribute name is truncated. For more information, see “Support for XML attributes” on page 107.
The nillable attribute has special rules. The prefix attr- is added, but nil- is also added to the beginning of the attribute name. The attribute name is followed by -value. If the total length is longer than 28 characters, the attribute name is truncated.

The total length of the resulting name is 30 characters or less.

DFHWS2LS maps schema types to COBOL data description elements using the specified mapping level according to the following table. You should also note the following points:

- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, variable length character data is mapped to null terminated strings.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to YES, variable length character data is mapped to two related elements - a length field and a data field. For example:

```
<xsd:simpleType name="VariableStringType">  
  <xsd:restriction base="xsd:string">  
    <xsd:minLength value="1"/>  
    <xsd:maxLength value="10000"/>  
  </xsd:restriction>  
</xsd:simpleType>  
<xsd:element name="textString" type="tns:VariableStringType"/>
```

maps to

15 textString-length PIC S9999 COMP-5 SYNC
 15 textString PIC X(10000)

Schema simple type	COBOL data description at mapping levels 1.0 and 1.1	COBOL data description at mapping level 1.2
<pre><xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType></pre>	Not supported	Not supported
<pre><xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType></pre>	PIC X(255) Supported at mapping level 1.1	PIC X(255)
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY hexBinary 	PIC X(z)	PIC X(z)
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> duration date dateTime time gDay gMonth gYear gMonthDay gYearMonth 	PIC X(32)	PIC X(32)
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> byte unsignedByte 	PIC X DISPLAY	PIC X DISPLAY

Schema simple type	COBOL data description at mapping levels 1.0 and 1.1	COBOL data description at mapping level 1.2
<code><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></code>	PIC S9999 COMP-5 SYNC or PIC S9999 DISPLAY	PIC S9999 COMP-5 SYNC or PIC S9999 DISPLAY
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></code>	PIC 9999 COMP-5 SYNC or PIC 9999 DISPLAY	PIC 9999 COMP-5 SYNC or PIC 9999 DISPLAY
<code><xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType></code>	PIC S9(18) COMP-3	PIC S9(18) COMP-3
<code><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></code>	PIC S9(9) COMP-5 SYNC or PIC S9(9) DISPLAY	PIC S9(9) COMP-5 SYNC or PIC S9(9) DISPLAY
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></code>	PIC 9(9) COMP-5 SYNC or PIC 9(9) DISPLAY	PIC 9(9) COMP-5 SYNC or PIC 9(9) DISPLAY
<code><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></code>	PIC S9(18) COMP-5 SYNC or PIC S9(18) DISPLAY	PIC S9(18) COMP-5 SYNC or PIC S9(18) DISPLAY
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></code>	PIC 9(18) COMP-5 SYNC or PIC 9(18) DISPLAY	PIC 9(18) COMP-5 SYNC or PIC 9(18) DISPLAY
<code><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="m"> <xsd:fractionDigits value="n"> </xsd:restriction> </xsd:simpleType></code>	PIC 9(p)V9(n) COMP-3 where $p = m - n$.	PIC 9(p)V9(n) COMP-3 where $p = m - n$.
<code><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></code>	PIC X DISPLAY	PIC X DISPLAY
<code><xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType></code>	PIC X(255) Supported at mapping level 1.1	PIC X(255)
<code><xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType></code>	PIC X(255) Supported at mapping level 1.1	PIC X(255)

Schema simple type	COBOL data description at mapping levels 1.0 and 1.1	COBOL data description at mapping level 1.2
<pre><xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType></pre> <p>where the length is not defined</p>	<p>PIC X(<i>y</i>)</p> <p>where $y = 4 \times (\text{ceil}(z/3))$. $\text{ceil}(x)$ is the smallest integer greater than or equal to x</p> <p>Supported at mapping level 1.1</p>	<p>PIC X(<i>z</i>)</p> <p>where the length is fixed</p> <p>PIC X(16)</p> <p>where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.</p>
<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>	PIC X(32)	COMP-1
<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>	PIC X(32)	COMP-2

Some of the Schema types shown in the table map to a COBOL format of COMP-5 SYNC or of DISPLAY, depending upon what values (if any) are specified in the <minInclusive> and <maxInclusive> facets:

- For signed types (short, int, and long), DISPLAY is used when the following are specified:

```
<xsd:minInclusive value="-a"/>
<xsd:maxInclusive value="a"/>
```

where *a* is a string of 9s.

- For unsigned types (unsignedShort, unsignedInt, and unsignedLong), DISPLAY is used when the following are specified:

```
<xsd:minInclusive value="0"/>
<xsd:maxInclusive value="a"/>
```

where *a* is a string of 9s.

When any other value is specified, or no value is specified, COMP-5 SYNC is used.

C/C++ and XML Schema mapping

Utility programs DFHLS2WS and DFHWS2LS support mappings between C and C++ data types and the XML Schema definitions that are included in each Web service description.

C and C++ to XML schema

C and C++ names are converted to XML names according to the following rules:

1. Characters that are not valid in XML element names are replaced with 'X'.
For example, monthly-total becomes monthlyXtotal.
2. Duplicate names are made unique by the addition of one or more numeric digits.
For example, two instances of year become year and year1.

DFHLS2WS maps C and C++ data types to schema elements according to the following table. C and C++ types that are not shown in the table are not supported by DFHLS2WS. The following restrictions also apply:

- Header files must contain a top level struct instance.
- You cannot declare a structure type that contains itself as a member
- The following C and C++ data types are not supported:
 - decimal
 - long double
 - wchar_t (C++ only)
- The following are ignored if they are present in the header file.

Storage class specifiers:

- auto
- register
- static
- extern
- mutable

Qualifiers

- const
- volatile
- _Export (C++ only)
- _Packed (C only)

Function specifiers

- inline (C++ only)
- virtual (C++ only)

Initial values

- The header file must not contain the following:
 - Unions
 - Class declarations
 - Enumeration data types
 - Pointer type variables
 - Template declarations
 - Predefined macros - that is, macros with names which start and end with two underscore characters (__)
 - The line continuation sequence (a \ symbol that is immediately followed by a newline character)
 - Prototype function declarators
 - Preprocessor directives
 - Bit fields
 - The __cdecl (or _cdecl) keyword (C++ only)
- The application programmer is required to use a 32 bit compiler to ensure that an int will map to 4 bytes.
- The following C++ reserved keywords are not supported:
 - explicit
 - using
 - namespace
 - typename
 - typeid

C and C++ data type	Schema simpleType
char[z]	<code><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></code>
char	<code><xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType></code>
unsigned char	<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType></code>
short	<code><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></code>
unsigned short	<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></code>
int long	<code><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></code>
unsigned int unsigned long	<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></code>
long long	<code><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></code>
unsigned long long	<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></code>
bool (C++ only)	<code><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></code>
float Supported at mapping level 1.2	<code><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></code>
double Supported at mapping level 1.2	<code><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></code>

XML Schema to C and C++

The CICS Web services assistant generates unique and valid names for C and C++ variables from the schema element names using the following rules:

1. Characters other than A-Z, a-z, 0-9, or _ are replaced with 'X'.

For example, monthly-total becomes monthlyXtotal.

2. If the first character is not an alphabetic character a leading 'X' is inserted.
For example, 6monthlysummary becomes X6monthlysummary.
3. If the schema element name is longer than 50 characters, it is truncated to that length.
4. Duplicate names in the same scope are made unique by the addition of two numeric digits.
For example, two instances of year become year and year1.
5. Five characters are reserved for the strings _cont or _num which are used when the schema specifies that the variable has varying cardinality; that is, when minOccurs and maxOccurs are specified on an xsd:element.
For more information, see "Variable arrays of elements" on page 104.
6. For attributes, the previous rules are applied to the element name. The prefix attr_ is added to the element name, and this is followed by _value or _exist. If the total length is longer than 28 characters, the element name is truncated.
The nillable attribute has special rules. The prefix attr_ is added, but nil_ is also added to the beginning of the element name. The element name is followed by _value. If the total length is longer than 28 characters, the element name is truncated.

The total length of the resulting name is 57 characters or less.

DFHWS2LS maps schema elements to C and C++ data types according to the following table. The following rules also apply:

- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, variable length character data is mapped to null terminated strings.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to YES, variable length character data is mapped to two related elements - a length field and a data field.

Schema simpleType	Mapping level 1.0 and 1.1	Mapping level 1.2
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	Not supported	Not supported
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	char[255] Supported at mapping level of 1.1	char[255]

Schema simpleType	Mapping level 1.0 and 1.1	Mapping level 1.2
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY hexBinary 	char[z]	char[z]
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> duration date dateTime time gDay gMonth gYear gMonthDay gYearMonth 	char[32]	char[32]
<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType></pre>	signed char	signed char
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType></pre>	char	char
<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>	short	short
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>	unsigned short	unsigned short

Schema simpleType	Mapping level 1.0 and 1.1	Mapping level 1.2
<code><xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType></code>	char[33]	char[33]
<code><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></code>	int	int
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></code>	unsigned int	unsigned int
<code><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></code>	long long	long long
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></code>	unsigned long long	unsigned long long
<code><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></code>	bool (C++ only) short (C only)	bool (C++ only) char (C only)
<code><xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType></code>	char[255] Supported at mapping level 1.1	char[255]
<code><xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType></code>	char[255] Supported at mapping level 1.1	char[255]
<code><xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64binary"> </xsd:restriction> </xsd:simpleType></code> where the length is not defined	char[y] where $y = 4 \times (\text{ceil}(z/3))$. ceil(x) is the smallest integer greater than or equal to x	char[z] where the length is fixed char[16] is the name of the container that stores the binary data when the length is not defined
<code><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></code>	char[32]	float(*)
<code><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></code>	char[32]	double(*)

PL/I and XML Schema mapping

Utility programs DFHLS2WS and DFHWS2LS support mappings between PL/I data structures and the XML Schema definitions that are included in each Web service description. Because there are differences between the Enterprise PL/I compiler and older PL/I compilers two language options are supported, PLI-ENTERPRISE and PLI-OTHER.

PL/I to XML Schema

PL/I names are converted to XML names according to the following rules:

1. Characters that are not valid in XML element names are replaced with 'x'.
For example, `monthly$total` becomes `monthlyxtotal`.
2. Duplicate names are made unique by the addition of one or more numeric digits.
For example, two instances of `year` become `year` and `year1`.

DFHLS2WS maps PL/I data types to schema elements according to the following table. PL/I types that are not shown in the table are not supported by DFHLS2WS. The following restrictions also apply:

- Data items with the COMPLEX attributes are not supported.
- Data items with the FLOAT attribute are supported at a mapping level of 1.2. Enterprise PL/I FLOAT IEEE is not supported.
- VARYING and VARYINGZ pure DBCS strings are supported at a mapping level of 1.2.
- Data items specified as `DECIMAL(p,q)` are supported only when $p \geq q$
- Data items specified as `BINARY(p,q)` are supported only when $q = 0$.
- If the PRECISION attribute is specified for a data item, it is ignored.
- PICTURE strings are not supported.
- ORDINAL data items are treated as `FIXED BINARY(7)` data types.

DFHLS2WS does not fully implement the padding algorithms of PL/I, and therefore you must declare padding bytes explicitly in your data structure. DFHLS2WS issues a message if it detects that padding bytes are missing. Each top level structure must start on a double word boundary and each byte within the structure must be mapped to the correct boundary. Consider this code fragment:

```
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

In this example:

- FIELD1 is 1 byte long and can be aligned on any boundary.
- FIELD2 is 4 bytes long and must be aligned on a full word boundary.
- FIELD3 is 8 bytes long and must be aligned on a double word boundary.

The Enterprise PL/I compiler aligns FIELD3 first, because it has the strongest boundary requirements. It then aligns FIELD2 at the fullword boundary immediately before FIELD3, and FIELD1 at the byte boundary immediately before FIELD3. Finally, so that the entire structure will be aligned at a fullword boundary, the compiler inserts three padding bytes immediately before FIELD1.

Because DFHLS2WS does not insert equivalent padding bytes, you must declare them explicitly before the structure is processed by DFHLS2WS. For example:

```

3 PAD1  FIXED BINARY(7),
3 PAD2  FIXED BINARY(7),
3 PAD3  FIXED BINARY(7),
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);

```

Alternatively, you can change the structure to declare all the fields as unaligned and recompile the application which uses the structure. For further information on PL/I structural memory alignment requirements refer to *Enterprise PL/I Language Reference*.

PL/I data description	Schema
FIXED BINARY (<i>n</i>) where $n \leq 7$	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
FIXED BINARY (<i>n</i>) where $8 \leq n \leq 15$	<xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType>
FIXED BINARY (<i>n</i>) where $16 \leq n \leq 31$	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>
FIXED BINARY (<i>n</i>) where $32 \leq n \leq 63$ Restriction: Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(<i>n</i>) where $n \leq 8$ Restriction: Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(<i>n</i>) where $9 \leq n \leq 16$ Restriction: Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(<i>n</i>) where $17 \leq n \leq 32$ Restriction: Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(<i>n</i>) where $33 \leq n \leq 64$ Restriction: Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType>
FIXED DECIMAL(<i>n,m</i>)	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value=" <i>n</i> "/> <xsd:fractionDigits value=" <i>m</i> "/> </xsd:restriction> </xsd:simpleType>
BIT(<i>n</i>) where <i>n</i> is a multiple of 8. Other values are not supported.	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value=" <i>m</i> "/> </xsd:restriction> </xsd:simpleType> where $m = n/8$

PL/I data description	Schema
CHARACTER(<i>n</i>)	<pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>
GRAPHIC(<i>n</i>)	<pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="m"/> </xsd:restriction> </xsd:simpleType></pre> <p>at a mapping level of 1.0 and 1.1, where $m = 2*n$</p> <pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre> <p>at a mapping level of 1.2.</p>
WIDECHAR(<i>n</i>)	<pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="m"/> </xsd:restriction> </xsd:simpleType></pre> <p>where $m = 2*n$</p> <pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="n"/> </xsd:restriction> </xsd:simpleType></pre> <p>at a mapping level of 1.2.</p>
ORDINAL Restriction: Enterprise PL/I only	<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType></pre>
BINARY FLOAT(<i>n</i>) where $n \leq 21$ Supported at mapping level 1.2.	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>
BINARY FLOAT(<i>n</i>) where $21 < n \leq 53$ Values greater than 53 are not supported. Supported at mapping level 1.2.	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>
DECIMAL FLOAT(<i>n</i>) where $n \leq 6$ Supported at mapping level 1.2.	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>
DECIMAL FLOAT(<i>n</i>) where $6 < n \leq 16$ Values greater than 16 are not supported. Supported at mapping level 1.2.	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>

XML Schema to PL/I

The CICS Web services assistant generates unique and valid names for PL/I variables from the schema element names using the following rules:

1. Characters other than A-Z, a-z, 0-9, @ # or \$ are replaced with 'X'.
For example, monthly-total becomes monthlyXtotal.
2. If the schema specifies that the variable has varying cardinality (that is, minOccurs and maxOccurs are specified with different values on the xsd:element), and the schema element name is longer than 24 characters, it is truncated to that length.
If the schema specifies that the variable has fixed cardinality, and the schema element name is longer than 29 characters, it is truncated to that length.
3. Duplicate names in the same scope are made unique by the addition of one or two numeric digits to the second and subsequent instances of the name.
For example, three instances of year become year, year1 and year2.
4. Five characters are reserved for the strings _cont or _num which are used when the schema specifies that the variable has varying cardinality; that is, when minOccurs and maxOccurs are specified with different values.
For more information, see “Variable arrays of elements” on page 104.
5. For attributes, the previous rules are applied to the attribute name. The prefix attr- is added to the attribute name, and this is followed by -value or -exist. If the total length is longer than 28 characters, the attribute name is truncated. For more information, see “Support for XML attributes” on page 107.
The nillable attribute has special rules. The prefix attr- is added, but nil- is also added to the beginning of the attribute name. The attribute name is followed by -value. If the total length is longer than 28 characters, the attribute name is truncated.

The total length of the resulting name is 31 characters or less.

DFHWS2LS maps schema elements to PL/I data types according to the following table. You should also note the following points:

- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is not specified, by default variable length character data is mapped to a VARYINGZ data type for Enterprise PL/I and VARYING data type for Other PL/I.
- Variable length binary data is mapped to a VARYING data type if it is less than 32,768 bytes and a container if it is more than 32,768 bytes.

Schema	PL/I data description at mapping level 1.0 and 1.1	PL/I data description at mapping level 1.2
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	Not supported	Not supported
<xsd:simpleType> <xsd:restriction base="xsd:anySimpleType"> </xsd:restriction> </xsd:simpleType>	CHAR(255) Supported at mapping level 1.1 or higher	CHAR(255)

Schema	PL/I data description at mapping level 1.0 and 1.1	PL/I data description at mapping level 1.2
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:maxLength value="z"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY 	CHARACTER(z)	CHARACTER(z)
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> duration date dateTime time gDay gMonth gYear gMonthDay gYearMonth 	CHAR(32)	CHAR(32)
<pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="y"/> </xsd:restriction> </xsd:simpleType></pre>	BIT(z) where $z = 8 \times y$ and $z < 4095$ bytes CHAR(z) where $z = 8 \times y$ and $z > 4095$ bytes.	CHAR(y)
<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I SIGNED FIXED BINARY (7) Other PL/I FIXED BINARY (7)	Enterprise PL/I SIGNED FIXED BINARY (7) Other PL/I FIXED BINARY (7)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I UNSIGNED FIXED BINARY (8) Other PL/I FIXED BINARY (8)	Enterprise PL/I UNSIGNED FIXED BINARY (8) Other PL/I FIXED BINARY (8)

Schema	PL/I data description at mapping level 1.0 and 1.1	PL/I data description at mapping level 1.2
<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I SIGNED FIXED BINARY (15) Other PL/I FIXED BINARY (15)	Enterprise PL/I SIGNED FIXED BINARY (15) Other PL/I FIXED BINARY (15)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I UNSIGNED FIXED BINARY (16) Other PL/I FIXED BINARY (16)	Enterprise PL/I UNSIGNED FIXED BINARY (16) Other PL/I FIXED BINARY (16)
<pre><xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I FIXED DECIMAL(31,0) Other PL/I FIXED DECIMAL(15,0)	Enterprise PL/I FIXED DECIMAL(31,0) Other PL/I FIXED DECIMAL(15,0)
<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I SIGNED FIXED BINARY (31) Other PL/I FIXED BINARY (31)	Enterprise PL/I SIGNED FIXED BINARY (31) Other PL/I FIXED BINARY (31)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I UNSIGNED FIXED BINARY(32) Other PL/I BIT(64)	Enterprise PL/I UNSIGNED FIXED BINARY(32) Other PL/I BIT(64)
<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I SIGNED FIXED BINARY(63) Other PL/I BIT(64)	Enterprise PL/I SIGNED FIXED BINARY(63) Other PL/I BIT(64)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></pre>	Enterprise PL/I UNSIGNED FIXED BINARY(64) Other PL/I BIT(64)	Enterprise PL/I UNSIGNED FIXED BINARY(64) Other PL/I BIT(64)

Schema	PL/I data description at mapping level 1.0 and 1.1	PL/I data description at mapping level 1.2
<pre><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></pre>	<p>Enterprise PL/I SIGNED FIXED BINARY (7)</p> <p>Other PL/I FIXED BINARY (7)</p>	<p>Enterprise PL/I BIT(7) BIT(1)</p> <p>Other PL/I BIT(7) BIT(1) where BIT(7) is provided for alignment and BIT(1) contains the boolean mapped value.</p>
<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="n"/> <xsd:fractionDigits value="m"/> </xsd:restriction> </xsd:simpleType></pre>	FIXED DECIMAL(<i>n,m</i>)	FIXED DECIMAL(<i>n,m</i>)
<pre><xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType></pre>	CHAR(255)	CHAR(255)
<pre><xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType></pre>	CHAR(255)	CHAR(255)
<pre><xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="y"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType></pre> <p>where the length is not defined</p>	<p>CHAR(<i>z</i>)</p> <p>where $z = 4 \times \text{ceil}(y/3)$. ceil(<i>x</i>) is the smallest integer greater than or equal to <i>x</i></p> <p>Supported at mapping level 1.1</p>	<p>CHAR(<i>y</i>)</p> <p>where the length is fixed CHAR(16)</p> <p>where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.</p>
<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>	CHAR(32)	<p>Enterprise PL/I DECIMAL FLOAT(6) HEXADEC</p> <p>Other PL/I DECIMAL FLOAT(6)</p>
<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>	CHAR(32)	<p>Enterprise PL/I DECIMAL FLOAT(16) HEXADEC</p> <p>Other PL/I DECIMAL FLOAT(16)</p>

Variable arrays of elements

A SOAP message can contain an array with varying numbers of elements. An array with a varying number of elements is represented in XML schema by using the `minOccurs` and `maxOccurs` attributes on the element declaration.

For example:

```
<xsd:element name="component" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

denotes an 8-byte string that is optional, that is, it can occur zero or one times in the SOAP message.

The following example denotes a 8-byte string that must occur at least once:

```
<xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

In general:

The `minOccurs` attribute specifies the minimum number of times the element can occur. It can have a value of 0 or any positive integer.

The `maxOccurs` attribute specifies the maximum number of time the element can occur. It can have a value of any positive integer greater than or equal to the value of the `minOccurs` attribute. It can also take a value of `unbounded`, which indicates that there is no upper limit to the number of times the element can occur.

The default value for both attributes is 1.

In general, SOAP messages that contain varying numbers of elements do not map efficiently into a single high-level language data structure. Therefore, to handle these cases, CICS uses a series of connected data structures that are passed to the application program in a series of containers. These structures are used as input and output from the application. When CICS receives a SOAP message, it is responsible for populating these structures and the application is responsible for reading them. Where CICS is sending a SOAP message, the application is responsible for populating these structures and CICS is responsible for reading them.

The format of these data structures is best explained with a series of examples. These examples use an array of simple 8-byte fields. However, the model supports arrays of complex data types and arrays of data types that contain other arrays.

Fixed number of elements

The first example illustrates an element that occurs exactly three times:

```
<xsd:element name="component" minOccurs="3" maxOccurs="3">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
```

```

        <xsd:length value="8"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:element>

```

In this example, because the number of times that the element occurs is known in advance, it can be represented as a fixed length array in a simple COBOL declaration (or the equivalent in other languages):

```
05 component PIC X(8) OCCURS 3 TIMES
```

Single, optional element

The second example illustrates an optional element that, if it occurs, occurs once.

```

<xsd:element name="component" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

```

In this case, the main data structure does not contain a declaration of an array. Instead, it contains a declaration of two fields:

```
05 component-num PIC S9(9) COMP-4
05 component-cont PIC X(16)
```

At run time, the first field (component-num) contains the number of times (zero or one in this case) that the element appears in the SOAP message, and the second field (component-cont) contains the name of a container.

A second data structure contains the declaration of the element itself:

```
01 DFHWS-component
  02 component PIC X(8)
```

So, to process the data structure in your application program, you must examine the value of component-num. If it is zero, there is no component element in the message, and the contents of component-cont is undefined. If it is one, the component element is in the container named in component-cont. The contents of the container are mapped by the DFHWS-component data structure.

Varying number of elements

The third example illustrates a mandatory element that can occur from one to five times.

```

<xsd:element name="component" minOccurs="1" maxOccurs="5">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

```

The data structures are exactly the same as for a single, optional element. The main data structure contains:

```
05 component-num PIC S9(9) COMP-4
05 component-cont PIC X(16)
```

The second data structure contains:

```
01 DFHWS-component
  02 component PIC X(8)
```

Processing of the main data structure is similar to that in the previous example: you must examine the value of `component-num` (although this time it will contain a value in the range 1-5) to find out how many times the element occurs. The element contents are located in the container named in `component-cont`. The difference is that, this time, the container holds an array of elements, where each element is mapped by the `DFHWS-component` data structure.

Summary

The last two cases are very similar - in fact they are both examples of the same general model. The rules can be summarized:

- Where a varying array cannot be represented in the main data structure, each element of the array is represented in a second data structure.
- The main data structure contains the number of elements in the array, and the name of a container which holds the array of elements.
- Each element in the array is mapped by the secondary data structure associated with the array.

Note: If the SOAP message consists of a single recurring element, `DFHWS2LS` generates two language structures. The main language structure contains the number of elements in the array and the name of a container which holds the array of elements. The second language structure maps a single instance of the recurring element.

Nested variable arrays

Complex SOAP messages may contain nested arrays with optional elements or with varying numbers of elements at different levels. When this is the case, the structure described is extended beyond the two levels described in the previous examples.

This example illustrates an optional element (`<component2>`) nested in a mandatory element (`<component1>`) that can occur from one to five times.

```
<xsd:element name="component1" minOccurs="1" maxOccurs="5">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="component2" minOccurs="0" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:length value="8"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The top level data structure is exactly the same as in the previous examples:

```
05 component1-num PIC S9(9) COMP-4
05 component1-cont PIC X(16)
```

But the second data structure contains:

```
01 DFHWS-component1
  02 component2-num PIC S9(9) COMP-4
  02 component2-cont PIC X(16)
```

And a third level structure contains:

```
01 DFHWS-component2
  02 component2 PIC X(8)
```

The number of occurrences of the outermost element (<component1>) is in component1-num.

The container named in component1-cont contains an array with that number of instances of the second data structure (DFHWS-component1).

Each instance of component2-cont names a different container, each of which contains the data structure mapped by the third level structure (DFHWS-component2).

To illustrate this, consider the fragment of XML that matches the example:

```
<component1><component2>string1</component2></component1>
<component1><component2>string2</component2></component1>
<component1></component1>
```

There are three instances of <component1>. The first two each contain an instance of <component2>; the third instance does not.

In the top level data structure, component1-num contains a value of 3. In the container named in component1-cont are three instances of DFHWS-component1:

1. In the first, component2-num has a value of 1, and the container named in component2-cont holds *string1*.
2. In the second, component2-num has a value of 1, and the container named in component2-cont holds *string2*.
3. In the third, component2-num has a value of 0, and the contents of component2-cont is undefined.

In this instance, the complete data structure is represented by four containers in all:

- The root data structure in container DFHWS-DATA.
- The container named in component1-cont.
- Two containers named in the first two instances of component2-cont.

Support for XML attributes

XML schemas can specify attributes that are allowed or required in a SOAP message. The Web services assistant utility DFHWS2LS ignores XML attributes by default. To process XML attributes that are defined in the XML Schema, apply APAR PK15904 and then change the value of the **MAPPING-LEVEL** parameter to 1.1 or higher in DFHWS2LS.

Optional attributes

Attributes can be optional or required and can be associated with any element in the SOAP message. For every optional attribute defined in the schema, two fields are generated in the appropriate language structure.

1. An existence flag - this field is treated as a boolean data type and is typically one byte in length.

2. A value - this field is mapped in the same way as an equivalently typed XML element. For example, an attribute of type NMTOKEN is mapped in the same way as an XML element of type NMTOKEN.

The attribute existence and value fields appear in the generated language structure before the field for the element they are associated with. Unexpected attributes that appear in the instance document are ignored.

For example, consider the following schema attribute definition:

```
<xsd:attribute name="age" type="xsd:short" use="optional" />
```

This optional attribute would be mapped to the following COBOL structure:

```
05 attr-age-exist PIC X DISPLAY  
05 attr-age-value PIC S9999 COMP-5 SYNC
```

Runtime processing of optional attributes

When CICS receives and reads SOAP messages, the following runtime processing takes place for optional attributes:

- If the attribute is present, the existence flag is set and the value is mapped.
- If the attribute is not present, the existence flag is not set.
- If the attribute has a default value and is present, the value is mapped.
- If the attribute has a default value and is not present, the default value is mapped.

Optional attributes that have default values are treated as required attributes.

When CICS produces a SOAP message based on the contents of a COMMAREA or a container, the following runtime processing takes place:

- If the existence flag is set, the attribute is transformed and included in the message.
- If the existence flag is not set, the attribute is not included in the message.

Required attributes and runtime processing

For every attribute that is required, only the value field is generated in the appropriate language structure.

When CICS receives and reads SOAP messages at run time, if the attribute is present then the value is mapped. If the attribute is not present:

- As the provider, CICS generates a SOAP fault message indicating there is an error in the client's SOAP message.
- As the requester, CICS returns a conversion error resp2 code of 13 to the application.

When CICS produces a SOAP message based on the contents of a COMMAREA or container, the attribute is transformed and included in the message.

The nillable attribute

The nillable attribute is a special attribute that can appear on an `xsd:element` within an XML schema. It specifies that the `xsi:nil` attribute is valid for the element in a

SOAP message. If an element has the `xsi:nil` attribute specified, it indicates that the element is present but has no value, and therefore no content is associated with it.

If an XML schema has defined the `nillable` attribute as `true`, then it is mapped as a required attribute that takes a boolean value.

In runtime processing, when CICS receives a SOAP message and reads an `xsi:nil` attribute:

- The value of the attribute is `true` or `false`.
- If the value is `true`, the values of the element or nested elements within the scope of the `xsi:nil` attribute must be ignored by the application.

When CICS produces a SOAP message based on the contents of a `COMMAREA` or container for which the value for the `xsi:nil` attribute is `true`:

- The `xsi:nil` attribute is generated into the SOAP message.
- The value of the associated element is ignored.
- Any nested elements within the element are ignored.

Consider the following example XML schema, which could be part of a WSDL document:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="root" nillable="true">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element nillable="true" name="num" type="xsd:int" maxOccurs="3" minOccurs="3"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

An example of a partial SOAP message that conforms to this schema is:

```
<?xml version="1.0"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <num xsi:nil="true"/>
  <num>15</num>
  <num xsi:nil="true"/>
</root>
```

In COBOL, this SOAP message would map to:

```
05  root
10  attr-nil-root-value  PIC X DISPLAY
10  num                  OCCURS 3
15  num1                 PIC S9(9) COMP-5 SYNC
15  attr-nil-num-value  PIC X DISPLAY
10  filler               PIC X(3)
```

Chapter 8. Interfacing with service provider and requester applications

You must code your service provider and service requester applications (or wrapper programs) to interact with the Web services support in CICS. How you do this depends upon whether you are developing a service provider application or a service requester, and whether you are using the CICS Web services assistant to deploy your application.

How an application is invoked in a service provider

The way you invoke an application program (or a wrapper program) in a service provider depends upon whether or not you are using the Web services assistant to deploy your application.

How CICS invokes a service provider program deployed with the Web services assistant

When a service provider application that has been deployed using the CICS Web services assistant is invoked, CICS links to it with a COMMAREA or a channel.

You specify which sort of interface is used when you run JCL procedure DFHWS2LS or DFHLS2WS with the **PGMINT** parameter. If you specify a channel, you can name the container in the **CONTID** parameter.

- If the program is invoked with a COMMAREA interface, the COMMAREA contains the top level data structure that CICS created from the SOAP request.
- If the program is invoked with a channel interface, the top level data structure is passed to your program in the container that was specified in the **CONTID** parameter of DFHWS2LS or DFHLS2WS. If you did not specify the **CONTID** parameter, the data is passed in container DFHWS-DATA. The channel interface supports arrays with varying numbers of elements, which are represented as series of connected data structures in a series of containers. These containers will also be present.

When you code API commands to work with the containers, you do not need to specify the CHANNEL option, because all the containers are associated with the current channel (the channel that was passed to the program). If you need to know the name of the channel, use the EXEC CICS ASSIGN CHANNEL command.

When your program has processed the request, it must use the same mechanism to return the response: if the request was received in a COMMAREA, then the response must be returned in the COMMAREA; if the request was received in a container, the response must be returned in the same container.

If an error is encountered when the application program is issuing a response message, CICS rolls back all of the changes unless the application has performed a syncpoint.

If the Web service provided by your program is not designed to return a response, CICS will ignore anything in the COMMAREA or container when the program returns.

How CICS invokes other service provider programs

When a service provider application that has not been deployed with the CICS Web services assistant is invoked, CICS links to it with a channel.

The following containers are available containing the request:

Container name

Contents

DFHWS-BODY

For an inbound SOAP request when the pipeline includes a CICS-provided SOAP message handler, the contents of the SOAP body.

DFHREQUEST

The complete request (including the envelope for a SOAP request) received from the pipeline.
--

DFHWS-XMLNS

A list of name-value pairs that map namespace prefixes to namespaces for the XML content of the request.
--

DFHWS-SOAPACTION

The SOAPAction header associated with the SOAP message in container DFHWS-BODY.

The following containers that were available to message handlers in the pipeline are also available to your program, as well as any other containers that you created:

- DFHFUNCTION
- DFHWS-SOAPLEVEL
- DFHWS-URI
- DFHWS-TRANID
- DFHWS-USERID
- DFHWS-APPHANDLER
- DFH-SERVICEPLIST
- DFHHANDLERPLIST
- DFHWS-PIPELINE

When your program has processed the request, it returns the response in the following containers:

Container name

Contents

DFHRESPONSE

The complete response message to be passed to the pipeline. Use this container if you do not use SOAP for your messages, or if you want to build the complete SOAP message (including the envelope) in your program.
--

If you supply a SOAP body in container DFHWS-BODY, DFHRESPONSE should be empty. If you supply content in both DFHWS-BODY and DFHRESPONSE, CICS uses DFHRESPONSE.
--

DFHWS-BODY

For an outbound SOAP response, the contents of the SOAP body. Provide this container when the terminal handler of your pipeline is a CICS-provided SOAP message handler. The message handler will construct the full SOAP message containing the body.
--

You can use any of the other containers to pass information that your pipeline needs for processing the outbound response.

If the terminal handler of your pipeline is a CICS-provided SOAP message handler, and your Web service does not return a response, you must return container DFHNORESPONSE to indicate that there is no response. The contents of the container are unimportant, as the message handler checks only whether the container is present or not.

When you are not using a SOAP message handler, you indicate that there is no response by not returning container DFHRESPONSE.

When you code API commands to work with the containers, you do not need to specify the CHANNEL option, because all the containers are associated with the current channel (the channel that was passed to the program). If you need to know the name of the channel, use the EXEC CICS ASSIGN CHANNEL command.

Invoking a Web service from a CICS program

The way you invoke a Web service from an application program (or from a wrapper program) depends upon whether or not you are using the Web services assistant to deploy your application.

Invoking a Web service from an application deployed with the Web services assistant

A service requester application that is deployed with the Web services assistant uses the EXEC CICS INVOKE WEBSERVICE command to invoke a Web service. The request and response are mapped to a data structure in container DFHWS-DATA.

1. Create a channel and populate it with containers. At the minimum, container DFHWS-DATA must be present. It holds the top level data structure that CICS will convert into a SOAP request. If the SOAP request contains any arrays that have varying numbers of elements, they are represented as a series of connected data structures in a series of containers. These containers must also be present in the channel.
2. Invoke the target Web service. Use this command:

```
EXEC CICS INVOKE WEBSERVICE(webservice)  
                CHANNEL(userchannel)  
                OPERATION(operation)
```

where:

webservice is the name of the WEBSERVICE resource that defines the Web service to be invoked. The WEBSERVICE resource specifies the location of the Web service description, and the Web service binding file that CICS uses when it communicates with the Web service.

userchannel is the channel that holds container DFHWS-DATA and any other containers associated with the application's data structure.

operation is the name of the operation that is to be invoked in the target Web service.

You can also specify URI(*uri*) where *uri* is the URI of the Web service to be invoked. If this option is omitted, then the Web service binding file associated with the WEBSERVICE resource definition must include either a provider URI

(obtained from the Web service description by DFHWS2LS) or a provider application name (specified as a parameter to DFHWS2LS). If you specify this option, it is used in place of the URI or provider application name specified in the Web service binding file.

The provider application name in the Web service binding file associated with the WEBSERVICE resource is used to enable local optimization of the Web service request. If you use this optimization, the EXEC CICS INVOKE WEBSERVICE command is optimized to an EXEC CICS LINK command. This optimization has an effect on the behavior of the EXEC CICS INVOKE WEBSERVICE command when the Web service is not expected to send a response:

- When the optimization is not in effect, control returns from the EXEC CICS INVOKE WEBSERVICE command as soon as the request message is sent.
- When the optimization is in effect, control returns from the EXEC CICS INVOKE WEBSERVICE command only when the target program terminates.

When the Web service is expected to send a response, control returns from the command when the response is available.

You can use this optimization only if the service provider application (as well as the service requester application) was deployed with the Web services assistant.

3. If the command was successful, retrieve the response containers from the channel. At the minimum, container DFHWS-DATA will be present. It holds the top level data structure that CICS created from the SOAP response. If the response contains any arrays that have varying numbers of elements, they are represented as series of connected data structures in a series of containers. These containers will be present in the channel.
4. If the service requester receives a SOAP fault message from the invoked Web service, you need to decide if the application program should roll back any changes. If this occurs, an INVREQ error with a RESP2 value of 6 is returned to the application program. However, if optimization is in effect, the same transaction is used in both the requester and provider. If an error occurs in a locally optimized Web service provider, all of the work done by the transaction rolls back in both the provider and the requester. An INVREQ error is returned to the requester with a RESP2 value of 16.

Invoking a Web service from other applications

A service requester application that is not deployed with the Web services assistant links to program DFHPIRT to invoke a Web service. The request message and response message, and other information that CICS uses to process the request are held in a set of containers.

1. Create a channel and populate it with containers. Provide the following information in each container:

Container name

Contents

DFHWS-PIPELINE

The name of the PIPELINE resource used for the outbound request.

DFHWS-URI

The URI of the target Web service

DFHWS-BODY

For an outbound SOAP request, the contents of the SOAP body. Provide this container when the pipeline includes a CICS-provided

SOAP message handler. The message handler will construct the full SOAP message containing the body.

DFHREQUEST

The complete request message to be passed to the pipeline. Use this container if you do not use SOAP for your messages, or if you want to build the complete SOAP message (including the envelope) in your program.

If you supply a SOAP body in container DFHWS-BODY, DFHREQUEST should be empty. If you supply content in both DFHWS-BODY and DFHREQUEST, CICS uses DFHREQUEST.

DFHWS-XMLNS

A list of name-value pairs that map namespace prefixes to namespaces for the XML content of the request.

DFHWS-SOAPACTION

The SOAPAction header to be added to the SOAP message specified in container DFHWS-BODY.

2. Link to program DFHPIRT. Use this command:

```
EXEC CICS LINK PROGRAM(DFHPIRT) CHANNEL(userchannel)
```

where *userchannel* is the channel which holds your containers.

3. Retrieve the containers that contain the response from the same channel. The complete response is contained in the following containers:

Container name

Contents

DFHRESPONSE

The complete response (including the envelope for a SOAP response) received from the pipeline.

DFHWS-BODY

When the pipeline includes a CICS-provided SOAP message handler, the contents of the SOAP body.

DFHERROR

Error information from the pipeline.

Run time limitations for code generated by the Web services assistant

At run time, CICS is capable of transforming almost any valid SOAP message that conforms to the Web service description (WSDL) into the equivalent data structures. However, there are some limitations that you should be aware of when developing a service requester or service provider application using the Web services assistant batch jobs.

Code pages

CICS can support SOAP messages sent to it in any code page if there is an appropriate HTTP or MQ header identifying the code page. CICS converts the SOAP message to UTF-8 to process it in the pipeline, before transforming it to the code page required by the application program. To minimize the performance impact, it is recommended that you use the UTF-8 code page when sending SOAP messages to CICS. CICS always sends SOAP messages in UTF-8.

CICS can only transform SOAP messages if the code page used to convert data between the SOAP message and the application program is EBCDIC. Applications that expect data to be encoded in code pages such as UTF-8, ASCII and ISO8859-1 are unsupported. If you want to use DBCS characters within your data structures and SOAP messages, then you must specify a code page that supports DBCS. The EBCDIC code page that you select must also be supported by both Java and z/OS conversion services. z/OS conversion services must also be configured to support the conversion from the code page of the SOAP message to UTF-8.

To set an appropriate code page, you can either use the **LOCALCCSID** system initialization parameter or the optional **CCSID** parameter in the Web services assistant jobs. If you use the **CCSID** parameter, the value that you specify overrides the **LOCALCCSID** code page for that particular Web service. If you do not specify the **CCSID** parameter, the **LOCALCCSID** code page is used to convert the data and the Web service binding file is encoded in US EBCDIC (Cp037).

Containers

In service provider mode, if you specify that the **PGMINT** parameter has a value of CHANNEL, then the container that holds your application data must be written to and read from in binary mode. This container is DFHWS-DATA by default. The PUT CONTAINER command must either have the DATATYPE option set to BIT, or you must omit the FROMCCSID option so that BIT remains the default. For example, the following code explicitly states that the data in the container CUSTOMER-RECORD on the current channel should be written in binary mode.

```
EXEC CICS PUT CONTAINER (CUSTOMER-RECORD)
      FROM (CREC)
      DATATYPE(BIT)
```

Although the containers themselves are all in BIT mode, any text fields within the language structure that map this data must use an EBCDIC code page - the same code page as you have specified in the **LOCALCCSID** or **CCSID** parameter. If you are using DFHWS2LS to generate the Web service binding file, there could be many containers on the channel that hold parts of the complete data structure. If this is the case, then the text fields in each of these containers must be read from and written to using the same code page.

If the application program is populating containers that are going to be converted to SOAP messages, the application is responsible for ensuring that the containers have the correct amount of content. If a container holds less data than expected, CICS issues a conversion error.

If an application program uses the INVOKE WEBSERVICE command, then any containers it passes to CICS could potentially be reused and the data within them replaced. To avoid this, make sure that you have finished with the data when you are passing a channel to CICS that has previously been used for another purpose. If you have a provider mode Web service that is also a requester mode Web service, it is recommended that you use a different channel when using the INVOKE WEBSERVICE command, rather than using the default channel that it was originally attached to. If your application program is using the INVOKE WEBSERVICE command many times, it is recommended that you either use different channels on each call to CICS, or ensure that all the important data from the first request is saved before making the second request.

Conforming with the Web services description

A Web service description could describe some of the possible content of a SOAP message as optional. If this is the case, DFHWS2LS allocates fields within the generated language structure to indicate whether the content is present or not. At run time, CICS populates these fields accordingly. If a field, for example an existence flag or an occurrence field, indicates that the information is not present, the application program should not attempt to process the fields associated with that optional content.

If a SOAP message is missing some of its content when CICS transforms it, the equivalent fields within the data structures are not initialized when passed to the application program.

A Web service description can also specify the white space processing rules to use when reading a SOAP message, and CICS implements these rules at run time.

- If the value of the `xsd:whiteSpace` facet is `replace`, the white space characters such as “tab” and “carriage return” are replaced with spaces.
- If the value of the `xsd:whiteSpace` facet is `collapse`, any leading and trailing white space characters are removed when generating SOAP messages.

SOAP messages

CICS does not support SOAP message content derivation. For example, a SOAP message could use the `xsi:type` attribute to specify that an element has a particular type, together with an `xsi:schemaLocation` attribute to specify the location of the schema that describes the element. CICS does not support the capability of dynamically retrieving the schema and transforming the value of the element based on the content of the schema. CICS does support the `xsi:nil` attribute when the mapping level set in the Web services assistant is 1.1 or higher, but this is the only XML schema instance attribute that is supported.

DFHWS2LS might have to make assumptions about the maximum length or size of some values in the SOAP message. For example, if the XML schema does not specify a maximum length for an `xsd:string`, then DFHWS2LS assumes that the maximum length is 255 characters and generates a language structure accordingly. You can change this value by using the **DEFAULT-CHAR-MAXLENGTH** parameter in DFHWS2LS. At run time, if CICS encounters a SOAP message with a value that is larger than the space that has been allocated in the language structure, CICS issues a conversion error.

If CICS is the service provider, a SOAP fault message is returned to the requester. If CICS is the service requester, then an appropriate RESP2 code is returned from the INVOKE WEBSERVICE command.

Some characters have special meanings in XML, such as the `<` and `>` characters. If any of these special characters appear within a character array that is processed by CICS at run time, then it is replaced with the equivalent entity. The XML entities that are supported are:

Character	XML entity
&	&
<	<
>	>

Character	XML entity
"	"
'	'

CICS also supports the canonical forms of the numeric character references used for white space codes:

Character	XML entity
Tab		
Carriage return	

Line feed	

Note that this support does not extend to any pipeline handler programs that are invoked.

The null character (x'00') is invalid in any XML document. If a character type field that is provided by the application program contains the null character, CICS truncates the data at that point. This allows you to treat character arrays as null terminated strings. Character type fields generated by DFHWS2LS from base64Binary or hexBinary XML schema data types represent binary data and could contain null characters without truncation.

SOAP fault messages

If CICS is the service provider, and you want the application program to issue a SOAP fault message, use the SOAPFAULT CREATE command. In order to use this API command, you must specify that the Web services assistant **PGMINT** parameter has a value of CHANNEL. If you do not specify this value, and the application program invokes the SOAPFAULT CREATE command, CICS does not attempt to generate a SOAP response message.

Chapter 9. The pipeline configuration file

The configuration of a pipeline used to handle a Web service request is specified in an XML document, known as a *pipeline configuration file*.

The pipeline configuration file is stored in the z/OS UNIX System Services hierarchical file system (HFS), and its name is specified in the CONFIGFILE attribute of a PIPELINE resource definition. Use a suitable XML editor or text editor to work with your pipeline configuration files. When you work with configuration files, ensure that the character set encoding is US EBCDIC (Code page 037).

When CICS processes a Web service request, it uses a pipeline of one or more message handlers to handle the request. A pipeline is configured to provide aspects of the execution environment that apply to different categories of applications, such as support for Web Service Security, and Web Service transactions. Typically, a CICS region that has a large number of service provider or service requester applications will need several different pipeline configurations. However, where different applications have similar requirements, they can share the same pipeline configuration.

There are two kinds of pipeline configuration: one describes the configuration of a service provider pipeline; the other describes a service requester pipeline. Each is defined by its own schema, and each has a different root element.

Pipeline	Schema	Root element
Service provider	Provider.xsd	<provider_pipeline>
Service requester	Requester.xsd	<requester_pipeline>

Although many of the XML elements used are common to both kinds of pipeline configuration, others are used only in one or the other, so you cannot use the same configuration file for both a provider and requester.

Restriction: Namespace-qualified element names are not supported in the pipeline configuration file.

The immediate sub-elements of the <provider_pipeline> and <requester_pipeline> elements are:

- A <service> element, which specifies the message handlers that are invoked for every request. This element is mandatory when used within the <provider_pipeline> element, and optional within the <requester_pipeline> element.
- An optional <transport> element, which specifies message handlers that are selected at run time, based upon the resources that are being used for the message transport.
- For the <provider_pipeline> only, an <apphandler> element, which is used in some cases to specify the target application (or wrapper program) that provides the service.
- An optional <service_parameter_list> element, which contains the parameters that are available to the message handlers in the pipeline.

Associated with the pipeline configuration file is a PIPELINEPIPELINE resource. The attributes include CONFIGFILE, which specifies the name of the pipeline configuration file in HFS. When you install a PIPELINE definition, CICS reads the information that it needs in order to configure the pipeline from the file.

CICS supplies sample configuration files that you can use as a basis for developing your own. They are provided in library /usr/lpp/cicts/samples/pipelines.

File	Description
------	-------------

basicsoap11provider.xml	A pipeline definition for a service provider that uses the CICS-provided SOAP 1.1 handler, for use when the application has been deployed using the CICS Web services assistant.
--------------------------------	--

basicsoap11requester.xml	A pipeline definition for a service requester that uses the CICS-provided SOAP 1.1 handler, for use when the application has been deployed using the CICS Web services assistant.
---------------------------------	---

wsatprovider.xml	A pipeline definition that adds configuration information for Web Services transactions to basicsoap11provider.xml.
-------------------------	---

wsatrequester.xml	A pipeline definition that adds configuration information for Web Services transactions to basicsoap11requester.xml.
--------------------------	--

Example pipeline configuration file

This is a simple example of a configuration file for a service provider pipeline:

```
<?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/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The pipeline contains just one message handler, the CICS-supplied SOAP 1.1 message handler. The handler links to program DFHPITP.

- The <provider_pipeline> element is the root element of the pipeline configuration file for a service provider pipeline.
- The <service> element specifies the message handlers that are invoked for every request. In the example, there is just one message handler.
- The <terminal_handler> element contains the definition of the terminal message handler of the pipeline.
- The <cics_soap_1.1_handler> indicates that the terminal handler of the pipeline is the CICS-supplied handler program for SOAP 1.1 messages.
- The <apphandler> element specifies the name of the program to which the terminal handler of the pipeline will link by default. In this case, the program is DFHPITP, which is the CICS-supplied target program for applications deployed with the CICS Web services assistant. For programs that are not deployed with the Web services assistant, this is the name of the target application program.

Transport-related handlers

In the configuration file for each pipeline, you can specify more than one set of message handlers. At run time, CICS selects the message handlers that are called, based upon the resources that are being used for the message transport.

In a service provider, and in a service requester, you can specify that some message handlers should be called only when a particular transport (HTTP or MQ) is in use. For example, consider a Web service that you make available to your employees. Those who work at a company location access the service using the MQ transport on a secure internal network; however, employees working at a business partner location access the service using the HTTP transport over the internet. In this situation, you might want to use message handlers to encrypt parts of the message when the HTTP transport is used, because of the sensitive nature of the information.

In a service provider, inbound messages are associated with a named resource (a TCPIPService for the HTTP transport, a QUEUE for the MQ transport). You can specify that some message handlers should be called only when a particular resource is used for an inbound request.

To make this possible, the message handlers are specified in two distinct parts of the pipeline configuration file:

The service section

Specifies the message handlers that are called each time the pipeline executes.

The transport section

Specifies the message handlers that might or might not be called, depending upon the transport resources that are in use.

Remember: At run time, a message handler can choose to curtail the execution of the pipeline. Therefore, even if CICS decides that a particular message handler should be called based on what is in the pipeline configuration file, the decision might be overruled by an earlier message handler.

The message handlers that are specified within the transport section (the *transport-related handlers*) are organized into several lists. At run time, CICS selects the handlers in just one of these lists for execution, based on which transport resources are in use. If more than one list matches the transport resources that are being used, CICS uses the list that is most selective. The lists that are used in both service provider and service requester pipelines are:

<default_transport_handler_list>

This is the least selective list of transport-related handlers; the handlers specified in this list are called when none of the following lists matches the transport resources that are being used.

<default_http_transport_handler_list>

In a service requester pipeline, the handlers in this list are called when the HTTP transport is in use.

In a service provider pipeline, the handlers in this list are called when the HTTP transport is in use, and no <named_transport_entry> names the TCPIPService for the TCP/IP connection.

<default_mq_transport_handler_list>

In a service requester pipeline, the handlers in this list are called when the WebSphere MQ transport is in use.

In a service provider pipeline, the handlers in this list are called when the WebSphere MQ transport is in use, and no <named_transport_entry> names the message queue on which inbound messages are received.

The following list of message handlers is used only in the configuration file for a service provider pipeline:

<named_transport_entry>

As well as a list of handlers, the <named_transport_entry> specifies the name of a resource, and the transport type.

- For the HTTP transport, the handlers in this list are called when the resource name matches the name of the TCPIPSERVICE for the inbound TCP/IP connection.
- For the WebSphere MQ transport, the handlers in this list are called when the resource name matches the name of the message queue that receives the inbound message.

Example

This is an example of a <transport> element from the pipeline configuration file for a service provider pipeline:

```
<transport>

  <!-- HANDLER1 and HANDLER2 are the default transport handlers -->
  <default_transport_handler_list>
    <handler><program>HANDLER1</program><handler_parameter_list/></handler>
    <handler><program>HANDLER2</program><handler_parameter_list/></handler>
  </default_transport_handler_list>

  <!-- HANDLER3 overrides defaults for MQ transport -->
  <default_mq_transport_handler_list>
    <handler><program>HANDLER3</program><handler_parameter_list/></handler>
  </default_mq_transport_handler_list>

  <!-- HANDLER4 overrides defaults for http transport with TCPIPSERVICE(WS00) -->
  <named_transport_entry type="http">
    <name>WS00</name>
    <transport_handler_list>
      <handler><program>HANDLER4</program><handler_parameter_list/></handler>
    </transport_handler_list>
  </named_transport_entry>

</transport>
```

The effect of this definition is this:

- The <default_mq_transport_handler_list> ensures that messages that use the MQ transport are processed by handler HANDLER3.
- The <named_transport_entry> ensures that messages that use the TCP/IP connection associated with TCPIPSERVICE(WS00) are processed by handler HANDLER4.
- The <default_transport_handler_list> ensures that all remaining messages, that is, those that use the HTTP transport, but not TCPIPSERVICE(WS00), are processed by handlers HANDLER1 and HANDLER2.

Remember: Any handlers specified in the service section of the pipeline definition will be called in addition to those specified in the transport section.

The pipeline definition for a service provider

The message handlers are defined in an XML document, which is stored in the HFS. The name of the file that contains the document is specified in the CFGFILE attribute of a PIPELINE definition.

The root element of the pipeline configuration document is the <provider_pipeline> element. The high-level structure of the document is shown in Figure 21.

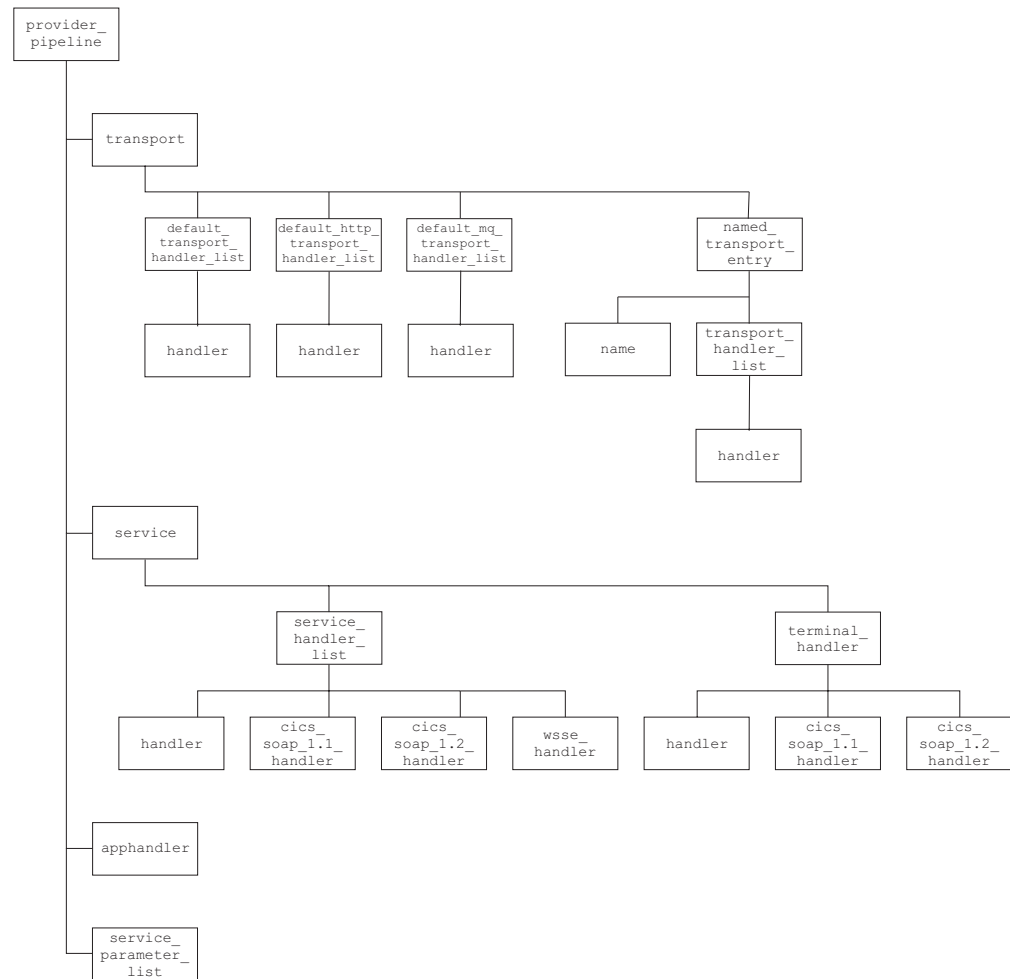


Figure 21. Structure of the pipeline definition for a service provider.

Note: In order to simplify the figure, child elements of the <handler>, <cics_soap_1.1_handler>, and <cics_soap_1.2_handler> elements are not shown.

The following elements are used only in the pipeline configuration for a service provider:

- <named_transport_entry>
- <terminal_handler>

Other elements are common to a service provider and a service requester.

The pipeline definition for a service requester

The message handlers are defined in an XML document, which is stored in the HFS. The name of the file that contains the document is specified in the CFGFILE attribute of a PIPELINE definition.

The root element of the pipeline configuration document is the <requester_pipeline> element. The high-level structure of the document is shown

in Figure 22.

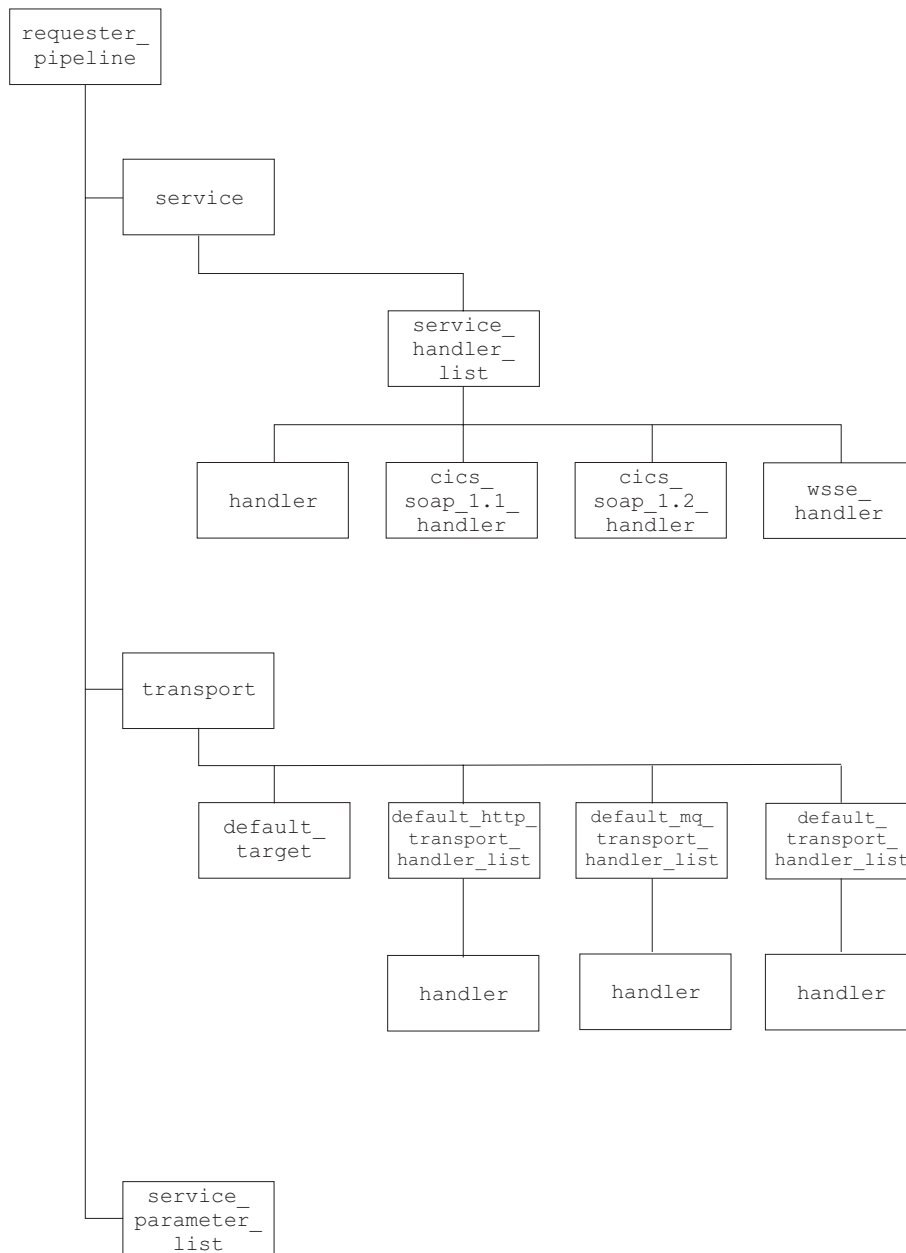


Figure 22. Structure of the pipeline definition for a service requester.

Note: In order to simplify the figure, child elements of the <handler>, <cics_soap_1.1_handler>, and <cics_soap_1.2_handler> elements are not shown.

Some elements used in the pipeline configuration for a service provider are also used in a service requester.

Elements used only in service providers

Some of the XML elements used in a pipeline configuration file apply only to service provider pipelines.

The <named_transport_entry> element

Contains a list of handlers that are to be invoked when a named transport resource is being used by a service provider.

- For the MQ transport, the named resource is the local input queue on which the request is received.
- For the HTTP transport, the resource is the TCPIP SERVICE that defines the port on which the request was received.

Used in:

- Service provider

Contained by:

<transport>

Attributes:

Name	Description
type	The transport mechanism with which the named resource is associated: wmq The named resource is a queue http The named resource is a TCPIP SERVICE

Contains:

1. A <name> element, containing the name of the resource
2. An optional <transport_handler_list> element. Each <transport_handler_list> contains one or more <handler> elements.
If you do not code a <transport_handler_list> element, then the only message handlers that are invoked when the named transport is used are those that are specified in the <service> element.

Example

```
<named_transport_entry type="http">  
  <name>PORT80</name>  
  <transport_handler_list>  
    <handler><program>HANDLER1</program><handler_parameter_list/></handler>  
    <handler><program>HANDLER2</program><handler_parameter_list/></handler>  
  </transport_handler_list>  
</named_transport_entry>
```

In this example, the message handlers specified (HANDLER1 and HANDLER2) are invoked for messages received on the TCPIP SERVICE with the name PORT80.

The <provider_pipeline> element

The root element of the XML document that describes the configuration of the CICS pipeline for a Web service provider.

Used in:

- Service provider

Contains:

1. Optional <transport> element
2. <service> element

3. Optional `<apphandler>` element, that specifies the name of the program that the terminal handler of the pipeline will link to by default.

Use the `<apphandler>` when the terminal handler is one of the CICS-supplied SOAP message handlers, that is when the `<terminal_handler>` element contains a `<cics_soap_1.1_handler>` element or a `<cics_soap_1.2_handler>` element.

Message handlers can specify a different program at run time, so the name coded here is not always the program that is linked to. If you do not code an `<apphandler>` element, one of the message handlers must use the DFHWS-APPHANDLER container to specify the name of the program at run time.

Important: When you use the CICS Web services assistant to deploy your service provider, the `<apphandler>` element (or the DFHWS-APPHANDLER container) must specify DFHPITP, and not the name of your target application or wrapper program. In this case, you specify the name of your program in the PGMNAME parameter when you run DFHWS2LS or DFHLS2WS.

4. Optional `<service_parameter_list>` element, containing XML elements that are made available to all the message handlers in the pipeline in container DFH-SERVICEPLIST.

Example

```
<provider_pipeline>
  <service>
    ...
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The `<terminal_handler>` element

Contains the definition of the terminal message handler of the service provider pipeline.

Used in:

- Service provider

Contained by:

- `<service>` element

Contains:

One of the following elements:

```
<handler>
<cics_soap_1.1_handler>
<cics_soap_1.2_handler>
```

However, you should not define `<cics_soap_1.1_handler>` and `<cics_soap_1.2_handler>` elements in the same pipeline. If you expect your pipeline to process both SOAP 1.1 and SOAP 1.2 messages, you should use the CICS-supplied SOAP 1.2 message handler.

Remember: In a service provider, you can specify these handlers in the `<service_handler_list>` element as well as in the `<terminal_handler>` element.

Example

```
<terminal_handler>
  <cics_soap_1.1_handler>
    ...
  </cics_soap_1.1_handler>
</service_handler_list>
```

The `<transport_handler_list>` element

Contains a list of message handlers that are invoked when a named resource is used.

- For the MQ transport, the named resource is the name of the local input queue.
- For the HTTP transport, the resource is the TCPIP SERVICE that defines the port on which the request was received.

Used in:

- Service provider

Contained by:

- `<named_transport_entry>` element

Contains:

- One or more `<handler>` elements.

Example

```
<transport_handler_list>
  <handler>
    ...
  </handler>
  <handler>
    ...
  </handler>
</transport_handler_list>
```

Elements used in service requesters

Some of the XML elements used in a pipeline configuration file apply only to service requester pipelines.

The `<requester_pipeline>` element

The root element of the XML document that describes the configuration of a pipeline in a service requester.

Used in:

- Service requester

Contains:

1. Optional `<service>` element
2. Optional `<transport>` element
3. Optional `<service_parameter_list>` element, containing XML elements that are made available to the message handlers in container DFH-SERVICEPLIST.

Example

```
<requester_pipeline>
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
    </service_handler_list>
  </service>
</requester_pipeline>
```

Elements used in service provider and requesters

Some of the XML elements used in a pipeline configuration file apply to both service provider and service requester pipelines.

The `<cics_soap_1.1_handler>` element

Defines the attributes of the CICS-supplied handler program for SOAP 1.1 messages.

Used in:

- Service requester
- Service provider

Contained by:

`<service_handler_list>` element
`<terminal_handler>` element

Contains:

Zero, one, or more `<headerprogram>` elements. Each `<headerprogram>` contains:

1. A `<program_name>` element, containing the name of a header processing program
2. A `<namespace>` element, which is used with the following `<localname>` element to determine which header blocks in a SOAP message should be processed by the header processing program. The `<namespace>` element contains the URI (Universal Resource Identifier) of the header block's namespace.
3. A `<localname>` element, which is used with the preceding `<namespace>` element 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.

For example, consider this header block:

```
<t:myheaderblock xmlns:t="http://mynamespace" ...> .... </t:myheaderblock>
```

- The namespace name is `http://mynamespace`
- The element name is `myheaderblock`

To make a header program match this header block, code the `<namespace>` and `<localname>` elements like this:

```
<namespace>http://mynamespace</namespace>
<localname>myheaderblock</localname>
```

You can code an asterisk (*) in the `<localname>` element to indicate that all header blocks in the namespace whose names begin with a given character string should be processed. For example:

```
<namespace>http://mynamespace</namespace>
<localname>myhead*</localname>
```

When you use the asterisk in the `<localname>` element, a header in a message can match more than one `<headerprogram>` element. For example, this header block

```
<t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
```

matches all the following `<headerprogram>` elements:

```
<headerprogram>
  <program_name>HDRPROG1</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>*</localname>
  <mandatory>>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG2</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myhead*</localname>
  <mandatory>>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG3</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myheaderblock</localname>
  <mandatory>>false</mandatory>
</headerprogram>
```

When this is the case, the header program that runs is the one specified in the `<headerprogram>` element in which the element name of the header block is most precisely stated. In the example, that is HDRPROG3.

When the SOAP message contains more than one header, the header processing program is invoked once for each matching header, but the sequence in which the headers are processed is undefined.

If you code two or more `<headerprogram>` elements that contain the same `<namespace>` and `<localname>`, but that specify different header programs, only one of the header programs will be called to process the header. The header will be passed in the DFHHEADER container to the selected program. The other header programs will not be called unless they are defined with `<mandatory>>true</mandatory>` in which case they will be called without having the header passed in the DFHHEADER container.

4. A `<mandatory>` element, containing an XML boolean value (true or false). Alternatively, you can code the values as 1 or 0 respectively.

true

During service request processing in a service provider pipeline, and service 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 messages 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.

During service request processing in a service requester pipeline, and service 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>`.

false

The header processing program is to be invoked only if one or more of the headers in the SOAP messages matches the <namespace> and <localname> elements:

- If none of the headers matches, the header processing program is not invoked.
- If any of the headers match, the header processing program is invoked once for each matching header.

Example

```
<cics_soap_1.1_handler>
  <headerprogram>
    <program_name> ... </program_name>
    <namespace>...</namespace>
    <localname>...</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.1_handler>
```

The <cics_soap_1.2_handler> element

Defines the attributes of the CICS-supplied SOAP 1.2 message handler program.

Used in:

- Service requester
- Service provider

Contained by:

<service_handler_list> element
<terminal_handler> element

Contains:

Zero, one, or more <headerprogram> elements. Each <headerprogram> contains:

1. A <program_name> element, containing the name of a header processing program
2. A <namespace> element, which is used with the following <localname> element to determine which header blocks in a SOAP message should be processed by the header processing program. The <namespace> element contains the URI (Universal Resource Identifier) of the header block's namespace.
3. A <localname> element, which is used with the preceding <namespace> element 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.

For example, consider this header block:

```
<t:myheaderblock xmlns:t="http://mynamespace" ...> .... </t:myheaderblock>
```

- The namespace name is http://mynamespace
- The element name is myheaderblock

To make a header program match this header block, code the <namespace> and <localname> elements like this:

```
<namespace>http://mynamespace</namespace>
<localname>myheaderblock</localname>
```

You can code an asterisk (*) in the <localname> element to indicate that all header blocks in the namespace whose names begin with a given character string should be processed. For example:

```
<namespace>http://mynamespace</namespace>
<localname>myhead*</localname>
```

When you use the asterisk in the <localname> element, a header in a message can match more than one <headerprogram> element. For example, this header block

```
<t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
```

matches all the following <headerprogram> elements:

```
<headerprogram>
  <program_name>HDRPROG1</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>*</localname>
  <mandatory>>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG2</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myhead*</localname>
  <mandatory>>false</mandatory>
</headerprogram>
<headerprogram>
  <program_name>HDRPROG3</program_name>
  <namespace>http://mynamespace</namespace>
  <localname>myheaderblock</localname>
  <mandatory>>false</mandatory>
</headerprogram>
```

When this is the case, the header program that runs is the one specified in the <headerprogram> element in which the element name of the header block is most precisely stated. In the example, that is HDRPROG3.

When the SOAP message contains more than one header, the header processing program is invoked once for each matching header, but the sequence in which the headers are processed is undefined.

If you code two or more <headerprogram> elements that contain the same <namespace> and <localname>, but that specify different header programs, only one of the header programs will be called to process the header. The header will be passed in the DFHHEADER container to the selected program. The other header programs will not be called unless they are defined with <mandatory>true</mandatory> in which case they will be called without having the header passed in the DFHHEADER container.

4. A <mandatory> element, containing an XML boolean value (true or false). Alternatively, you can code the values as 1 or 0 respectively.

true

During service request processing in a service provider pipeline, and service 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 messages 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.

During service request processing in a service requester pipeline, and service 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>`.

false

The header processing program is to be invoked only if one or more of the headers in the SOAP messages matches the `<namespace>` and `<localname>` elements:

- If none of the headers matches, the header processing program is not invoked.
- If any of the headers match, the header processing program is invoked once for each matching header.

Example

```
<cics_soap_1.2_handler>
  <headerprogram>
    <program_name> ... </program_name>
    <namespace>...</namespace>
    <localname>...</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler>
```

The `<default_http_transport_handler_list>` element

Specifies the message handlers that are invoked by default when the HTTP transport is in use.

In a service provider, message handlers specified in this list are invoked only if the list of handlers defined in a `<named_transport_entry>` element is less specific.

Used in:

- Service provider
- Service requester

Contained by:

- `<transport>` element

Contains:

- One or more `<handler>` elements.

Example

```
<default_http_transport_handler_list>
  <handler>
    ...
  </handler>
  <handler>
    ...
  </handler>
</default_http_transport_handler_list>
```


The <default_mq_transport_handler_list> element

Specifies the message handlers that are invoked by default when the MQ transport is in use.

In a service provider, message handlers specified in this list are invoked only if the list of handlers defined in a <named_transport_entry> element is less specific.

Used in:

- Service provider
- Service requester

Contained by:

- <transport> element

Contains:

- One or more <handler> elements.

Example

```
<default_mq_transport_handler_list>
  <handler>
    ...
  </handler>
  <handler>
    ...
  </handler>
</default_mq_transport_handler_list>
```

The <default_transport_handler_list> element

Specifies the message handlers that are invoked by default when any transport is in use.

In a service provider, message handlers specified in this list are invoked when the list of handlers defined in any of the following elements is less specific:

```
<default_http_transport_handler_list>
<default_mq_transport_handler_list>
<named_transport_entry>
```

Used in:

- Service provider
- Service requester

Contained by:

- <transport> element

Contains:

- One or more <handler> elements.

Example

```
<default_transport_handler_list>
  <handler><program>HANDLER1</program></handler>
  <handler><program>HANDLER2</program></handler>
</default_transport_handler_list>
```

The <handler> element

Defines the attributes of a message handler program.

Some CICS-supplied handler programs do not use the <handler> element. For example, the CICS-supplied SOAP message handler programs are defined using the <cics_soap_1.1_handler> and <cics_soap_1.2_handler> elements.

Used in:

- Service provider
- Service requester

Contained by:

```
<default_transport_handler_list>  
<transport_handler_list>  
<service_handler_list>  
<terminal_handler>  
<default_http_transport_handler_list>  
<default_mq_transport_handler_list>
```

Contains:

1. <program> element, containing the name of the handler program
2. <handler_parameter_list> element, containing XML elements that are made available to the message handlers in container DFH-HANDLERPLIST.

Example

```
<handler>  
  <program>MYPROG</program>  
  <handler_parameter_list><output print="yes"/></handler_parameter_list>  
</handler>
```

In this example, the handler program is MYPROG. The handler parameter list consists of a single <output> element; the contents of the parameter list are known to MYPROG.

The <service> element

Specifies the message handlers that are invoked for every request.

Used in:

- Service provider
- Service requester

Contained by:

```
<provider_pipeline>  
<requester_pipeline>
```

Contains:

1. <service_handler_list> element
2. In a service provider only, a <terminal_handler> element

Example

```
<service>
  <service_handler_list>
    ...
  </service_handler_list>
  <terminal_handler>
    ...
  </terminal_handler>
</service>
```

The <service_handler_list> element

Specifies a list of message handlers that are invoked for every request.

Used in:

- Service provider
- Service requester

Contained by:

- <service> element

Contains:

One or more of the following elements:

```
<handler>
<cics_soap_1.1_handler>
<cics_soap_1.2_handler>
<wsse_handler>
```

However, you should not define <cics_soap_1.1_handler> and <cics_soap_1.2_handler> elements in the same pipeline. If you expect your pipeline to process both SOAP 1.1 and SOAP 1.2 messages, you should use the CICS-supplied SOAP 1.2 message handler.

Remember: In a service provider, you can specify all of these handlers in the <terminal_handler> element, except for the <wsse_handler> handler element.

Example

In a service requester pipeline configuration file, you might specify:

```
<service_handler_list>
  <handler>
    ...
  </handler>
  <cics_soap_1.1_handler>
    ...
  </cics_soap_1.1_handler>
  <wsse_handler>
    ...
  </wsse_handler>
</service_handler_list>
```

The <service_parameter_list> element

An optional element containing XML elements that are made available to all the message handlers in the pipeline in container DFH-SERVICEPLIST.

Used in:

- Service requester
- Service provider

Contains:

- If you are using WS-AT: a <registration_service_endpoint> element
- In a service requester if you are using WS-AT: an optional <new_tx_context_required/> element
- Optional user defined tags

Example

```
<requester_pipeline>
  <service_parameter_list>
    <registration_service_endpoint>
      http://provider.example.com:7160/cicswsat/RegistrationService
    </registration_service_endpoint>
    <new_tx_context_required/>
    <user_defined_tag1>
      ...
    </user_defined_tag1>
  </service_parameter_list>
</requester_pipeline>
```

Related reference

“The <requester_pipeline> element” on page 127

The root element of the XML document that describes the configuration of a pipeline in a service requester.

“The <provider_pipeline> element” on page 125

The root element of the XML document that describes the configuration of the CICS pipeline for a Web service provider.

The <transport> element

Specifies handlers that are to be invoked only when a particular transport is in use.

Used in:

- Service provider
- Service requester

Contained by:

```
<provider_pipeline>
<requester_pipeline>
```

Contains:

In a service provider:

1. An optional <default_transport_handler_list> element
2. An optional <default_http_transport_handler_list> element
3. An optional <default_mq_transport_handler_list> element
4. Zero, one, or more <named_transport_entry> elements

In a service requester:

1. An optional <default_target> element. The <default_target> contains a URI that CICS uses to locate the target Web service when the service requester application does not provide a URI. In many cases, however, the URI of the

target will be provided by the service requester application, and whatever you specify in the <default_target> will be ignored. For example, service provider applications that are deployed using the CICS Web services assistant normally get the URI from the Web service description.

2. An optional <default_http_transport_handler_list> element
3. An optional <default_mq_transport_handler_list> element
4. An optional <default_transport_handler_list> element

Example

```
<transport>
  <default_transport_handler_list>
    ...
  </default_transport_handler_list>
</transport>
```

Pipeline configuration for WS-Security

In order for Web service requester and provider applications to participate in WS-Security protocols, you must configure your pipelines accordingly, by including message handler DFHWSSE1, and by providing configuration information for the handler.

The <wsse_handler> element

Specifies parameters used by the CICS-supplied message handler that provides support for WS-Security.

Used in:

- Service provider
- Service requester

Contained by:

<service_handler_list>

Contains:

- A <dfhwsse_configuration> element.

The <dfhwsse_configuration> element

Specifies configuration information for handler DFHWSSE1, which provides support for WS-Security.

Used in:

- Service provider
- Service requester

Contained by:

<wsse_handler>

Attributes:

Name	Description
version	An integer denoting the version of the configuration information. The only valid value is 1.

Contains:

1. An optional <authentication> element.
 - In a service requester pipeline, the <authentication> element specifies that CICS should add an X.509 certificate to the security header in outbound SOAP messages.
 - 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.
2. An optional, empty <expect_signed_body/> element.

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.
3. An optional, empty <expect_encrypted_body/> element.

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.
4. An optional <sign_body> element.

If this element is present, CICS will sign the <body> of the outbound message, using the algorithm specified in the <algorithm> element contained in the <sign_body> element.
5. An optional <encrypt_body> element.

If this element is present, CICS will encrypt the <body> of the outbound message, using the algorithm specified in the <algorithm> element contained in the <encrypt_body> element.

Example

```
<dfhwsse_configuration version="1">
  <authentication mode="basic">
    <certificate_label>AUTHCERT03</certificate_label>
    <suppress/>
    <algorithm>http://www.w3.org/2000/09/xmldsig#dsa-sha1</algorithm>
  </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>
```

The <authentication> element

Specifies the use of security tokens in the headers of inbound and outbound SOAP messages.

Used in:

- Service provider
- Service requester

Contained by:

<dfhwsse_configuration>

Attributes:

Attribute	Description
trust	<p>Taken together, the trust and mode attributes specify:</p> <ul style="list-style-type: none"> • whether asserted identity is used • the combination of security tokens that are used in SOAP messages. <p>Asserted identity allows a trusted user to assert that work should run under an different identity, the <i>asserted identity</i>, without the trusted user having the credentials associated with that identity.</p> <p>When asserted identity is used, messages contain a <i>trust token</i> and an <i>identity token</i>. 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, that is, the user ID under which the request is executed.</p>
mode	<p>Use of asserted identity requires that a 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.</p> <p>The allowable combinations of the these attributes, and their meanings, are described in Table 1 and Table 2 on page 140.</p>

Table 1. The **mode** and **trust** attributes in a service requester pipeline

trust	mode	Meaning
none	none	No credentials are added to the message
	basic	<i>Invalid combination of attribute values</i>
	signature	Asserted identity is not used. CICS uses a single X.509 security token which is added to the message, and used to sign the message body. The certificate is identified with the <certificate_label> element, and the algorithm is specified in the <algorithm> element.
basic	(any)	<i>Invalid combination of attribute values</i>
signature	none	<i>Invalid combination of attribute values</i>
	basic	<p>Asserted identity is used. CICS adds the following tokens to the message:</p> <ul style="list-style-type: none"> • The trust token is an X.509 security token. • The identity token is a username with no password. <p>The certificate used to sign the identity token and message body is specified by the <certificate_label>. The user ID placed in the identity token is the contents of the DFHWS-USERID container (which, by default, contains the running task's user ID).</p>
	signature	<i>Invalid combination of attribute values</i>

Table 2. The **mode** and **trust** attributes in a service provider pipeline

trust	mode	Meaning
none	none	Inbound messages need not contain any credentials, and CICS does not attempt to extract or verify any credentials that are found in a message. However, CICS will check that any signed elements have been correctly signed.
	basic	Inbound messages must contain a username security token with a password. CICS puts the username in container DFHWS-USERID.
	signature	Inbound messages must contain an X.509 security token.
basic	none	<i>Invalid combination of attribute values</i>
	basic	Inbound messages must use asserted identity: <ul style="list-style-type: none"> The trust token is a username token with a password The identity token is a second username token without a password. CICS puts this username in container DFHWS-USERID, and this user ID is used to run transactions in CICS.
	signature	Inbound messages must use asserted identity: <ul style="list-style-type: none"> The trust token is a username token with a password The identity token is an X.509 certificate. CICS puts the user ID associated with the certificate in container DFHWS-USERID, and this user ID is used to run transactions in CICS.
signature	none	<i>Invalid combination of attribute values</i>
	basic	Inbound messages must use asserted identity: <ul style="list-style-type: none"> The trust token is an X.509 certificate The identity token is a username token without a password. CICS puts the username in container DFHWS-USERID, and this user ID is used to run transactions in CICS. <p>The identity token and the body must be signed with the X.509 certificate.</p>
	signature	Inbound messages must use asserted identity: <ul style="list-style-type: none"> The trust token is an X.509 certificate The identity token is a second X.509 certificate. CICS puts the user ID associated with this certificate in container DFHWS-USERID, and this user ID is used to run transactions in CICS. <p>The identity token and the body must be signed with the first X.509 certificate (the trust token).</p>

Notes:

1. The combinations of the trust and mode attribute values are checked when the PIPELINE resource is installed. The installation fails if the attributes are incorrectly coded.

Contains:

1. An optional <certificate_label> element that specifies the label associated with an X.509 digital certificate installed in RACF®. If this element is specified in

a service requester pipeline, and the <suppress> element is not specified, the certificate is added to the security header in the SOAP message. If you do not specify a <certificate_label> element, CICS uses the default certificate in the RACF key ring. The certificate label should not contain any of the following characters:

< > : ! =

This element is ignored in a service provider pipeline.

2. An optional, empty <suppress/> element.

If this element is specified in a service provider pipeline, the handler will not attempt to use any security tokens in the message to determine under which user ID the work will run.

If this element is specified in a service requester pipeline, the handler will not attempt to add to the outbound SOAP message any of the security tokens that are required for authentication.

3. An <algorithm> element that specifies the URI of the algorithm used to sign the body of the SOAP message.

You can specify the following algorithms:

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

The DSA with SHA1 signature algorithm is supported on inbound SOAP messages only. If you are using basic authentication on inbound SOAP messages, you must still specify the <algorithm> element.

Example

```
<authentication trust="signature" mode="basic">
  <certificate_label>AUTHCERT03</certificate_label>
  <suppress/>
  <algorithm>http://www.w3.org/2000/09/xmldsig#dsa-sha1</algorithm>
</authentication>
```

The <sign_body> element

Directs DFHWSSE1 to sign the body of outbound SOAP messages, and provides information about how the messages are to be signed.

Used in:

- Service provider
- Service requester

Contained by:

<dfhwsse_configuration>

Contains:

1. An <algorithm> element that contains the URI that identifies the algorithm used to sign the body of the SOAP message.

You can specify the following algorithms:

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

2. A <certificate_label> element that specifies the label associated with a digital certificate installed in RACF. This digital certificate should contain the private key, as this is used to sign the message. The public key associated with the private key is then sent in the SOAP message, allowing the signature to be validated.

Example

```
<sign_body>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>SIGCERT01</certificate_label>
</sign_body>
```

The <encrypt_body> element

Directs DFHWSSE1 to encrypt the body of outbound SOAP messages, and provides information about how the messages are to be encrypted.

Used in:

- Service provider
- Service requester

Contained by:

<dfhwsse_configuration>

Contains:

1. An <algorithm> element containing the URI that identifies the algorithm used to encrypt the body of the SOAP message.
You can specify the following algorithms:

Algorithm	URI
Triple Data Encryption Standard algorithm (Triple DES)	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	http://www.w3.org/2001/04/xmlenc#aes128-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	http://www.w3.org/2001/04/xmlenc#aes192-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	http://www.w3.org/2001/04/xmlenc#aes256-cbc

The public key that identifies the algorithm used to encrypt the message is sent in the SOAP message and encrypted using the key encryption algorithm RSA 1.5.

2. A <certificate_label> element that specifies the label that is associated with a digital certificate in RACF. 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

```
<encrypt_body>  
  <algorithm>http://www.w3.org/2001/04/xmlenc#aes256-cbc</algorithm>  
  <certificate_label>ENCCERT02</certificate_label>  
</encrypt_body>
```

Chapter 10. Message handlers

A message handler is a CICS program which is used to process a Web service request during input, and to process the response during output. Message handlers use channels and containers to interact with one another, and with the system.

The message handler interface lets you perform the following tasks in a message handler program:

- Examine the contents of an XML request or response, without changing it
- Change the contents of an XML request or response
- In a non-terminal message handler, pass an XML request or response to the next message handler in the pipeline
- In a terminal message handler, call an application program, and generate a response
- In the request phase of the pipeline, force a transition to the response phase, by absorbing the request, and generating a response
- Handle errors

Tip: It is advisable to use the CICS-provided SOAP 1.1 and SOAP 1.2 handlers to work with SOAP messages. These handlers let you work directly with the major elements in a SOAP message (the SOAP headers and the SOAP body).

All programs which are used as message handlers are invoked with the same interface: they are invoked with a *channel* which holds a number of containers. The containers can be categorized as:

Control containers

These are essential to the operation of the pipeline. Message handlers can use the control containers to modify the sequence in which subsequent handlers are processed.

Context containers

In some situations, message handler programs need information about the context in which they are invoked. CICS provides this information in a set of *context containers* which are passed to the programs.

Some of the context containers hold information which you can change in your message handler. For example, in a service provider pipeline, you can change the user ID and transaction ID of the target application program by modifying the contents of the appropriate context containers.

User containers

These contain information which one message handler needs to pass to another. The use of user containers is entirely a matter for the message handlers.

Restriction: Do not use names starting with DFH for user containers.

Message handler protocols

Message handlers in a pipeline process request and response messages. The behavior of the handlers is governed by a set of protocols which describe what actions the message handlers can take in a given situation.

Each non-terminal message handler in a pipeline is invoked twice:

1. The first time, it is driven to process a request (an inbound request for a service provider pipeline, an outbound request for a service requester)
2. The second time, it is driven for one of three reasons:
 - to process a response (an outbound response for a service provider pipeline, an inbound response for a service requester)
 - to perform recovery following an error elsewhere in the pipeline
 - to perform any further processing that is required when there is no response.

The terminal message handler in a service provider pipeline is invoked once, to process a request.

Message handlers may be provided in a pipeline for a variety of reasons, and the processing that each handler performs may be very different. In particular:

- Some message handlers do not change the message contents, nor do they change the normal processing sequence of a pipeline
- Some message handlers change the message contents, but do not change the normal processing sequence of a pipeline
- Some message handlers change the processing sequence of a pipeline.

Each handler has a choice of actions that it can perform. The choice depends upon:

- whether the handler is invoked in a service provider or a service requester
- in a service provider, whether the handler is a terminal handler or not
- whether the handler is invoked for a request or a response message.

Terminal handler protocols

Normal request and response

This is the normal protocol for a terminal handler. The handler is invoked for a request message, and constructs a response.



In order to construct the response, a typical terminal handler will link to the target application program, but this is not mandatory.

Normal request, with no response

This is another common protocol for a terminal handler.



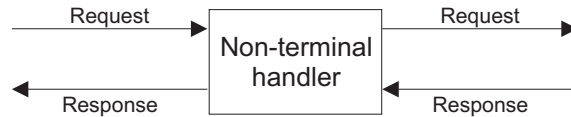
This protocol is usually encountered when the target application determines that there should be no response to the request (although the decision may also be made in the terminal handler).

Non-terminal handler protocols

Normal request and response

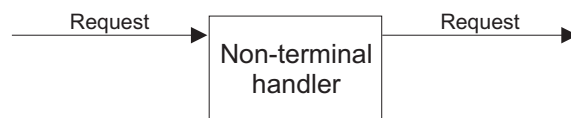
This is the usual protocol for a non-terminal handler. The handler is invoked for a request message, and again for the response message. In each case,

the handler processes the message, and passes it to the next handler in the pipeline.



Normal request, no response

This is another common protocol for a non-terminal handler. The handler is invoked for a request message, and after processing it, passes to the next handler in the pipeline. The target application (or another handler) determines that there should be no response. When the handler is invoked for the second time, there is no response message to process.



Handler creates the response

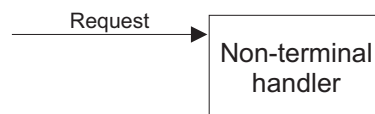
This protocol is typically used in abnormal situations, because the non-terminal handler does not pass the request to the next handler. Instead it constructs a response, and returns it to the pipeline.



This protocol could be used when the handler determines that the request is in some way invalid, and that no further processing of the request should be attempted. In this situation, the handler is **not** invoked a second time.

Handler suppresses the response

This is another protocol that is typically used in abnormal situations, because the non-terminal handler does not pass the request to the next handler. In this protocol, the handler determines that there should be no response to the request.



This protocol could be used when no response is expected to the original request, and, because the request is in some way invalid, no further processing of the request should be attempted.

Supplying your own message handlers

When you want to perform specialized processing on the messages that flow between a service requester and a service provider, and CICS does not supply a message handler that meets your needs, you will need to supply your own.

In most situations, you can perform all the processing you need with the CICS-supplied message handlers. For example, you can use the SOAP 1.1 and 1.2 message handlers which CICS supplies to process SOAP messages. But there are occasions when you will want to perform your own, specialized, operations on Web service requests and responses. To do this, you must supply your own message handlers.

1. Write your message handler program. A message handler is a CICS program with a channel interface. You can write your program in any of the languages which CICS supports, and use any CICS command in the DPL subset within your program.
2. Compile and link-edit your program. Message handler programs normally run under transaction CPIH, which is defined with the attribute TASKDATALOC(ANY). Therefore, when you link-edit the program, you must specify the AMODE(31) option.
3. Install the program in your CICS system in the usual way.
4. Define the program in the pipeline configuration file. Use the <handler> element to define your message handler. Within the <handler> element, code a <program> element containing the name of the program.

Working with messages in a non-terminal message handler

A typical non-terminal message handler processes a message, then passes control to another message handler in the pipeline.

In a non-terminal message handler, you can work with a request or response, with or without changing it, and pass it on to the next message handler.

Note: Although Web services typically use SOAP messages which contain XML, your message handlers will work as well with other message formats

1. Using the contents of container DFHFUNTION, determine if the message passed to this message handler is a request or a response.

DFHFUNTION	Request or response	Type of message handler	Inbound or outbound
RECEIVE-REQUEST	Request	Non-terminal	Inbound
SEND-RESPONSE	response	Non-terminal	Outbound
SEND-REQUEST	Request	Non-terminal	Outbound
RECEIVE-RESPONSE	response	Non-terminal	Inbound

Tip:

- If DFHFUNTION contains PROCESS-REQUEST, the message handler is a terminal message handler, and these steps do not apply.
 - If DFHFUNTION contains HANDLER-ERROR, the handler is being called for error processing, and these steps do not apply.
2. Retrieve the request or the response from the appropriate container.
 - If the message is a request, it is passed to the program in container DFHREQUEST. Container DFHRESPONSE is also present, with a length of zero.
 - If the message is a response, it is passed to the program in container DFHRESPONSE.

3. Perform any processing of the message which is required. Depending upon the purpose of the message handler, you might:
 - Examine the message without changing it, and pass it to the next message handler in the pipeline.
 - Change the request, and pass it to the next message handler in the pipeline.
 - If the message is a request, you can bypass the following message handlers in the pipeline, and, instead, construct a response message.

Note: It is the contents of the containers which a message handler returns that determines which message handler is invoked next. It is an error if a message handler does nothing (that is, it makes no changes to any of the containers passed to it).

Passing a message to the next message handler in the pipeline

In a typical non-terminal message handler, you will process a request or response, with or without changing it, and pass it on to the next message handler.

1. Return the message to the pipeline - changed or unchanged - in the appropriate container.
 - If the message is a request and you have changed it, return it in container DFHREQUEST
 - If the message is a response and you have changed it, put it in container DFHRESPONSE
 - If you have not changed the message, it is already in the appropriate container
2. If the message is a request, delete container DFHRESPONSE. When a message handler is invoked for a request, containers DFHREQUEST and DFHRESPONSE are passed to the program; DFHRESPONSE has a length of zero. However, it is an error to return both DFHREQUEST and DFHRESPONSE.

The message is passed to the next message handler on the pipeline.

Forcing a transition to the response phase of the pipeline

When you are processing a request, there are times when you will want to generate an immediate response, instead of passing the request to the next message handler in the pipeline.

1. Delete container DFHREQUEST.
2. Construct your response, and put it in container DFHRESPONSE.

The response is passed to the next message handler on the response phase of the pipeline.

Suppressing the response

In some situations, you will want to absorb a request without sending a response.

1. Delete container DFHREQUEST.
2. Delete container DFHRESPONSE.

Handling one way messages in a service requester pipeline

When a service requester pipeline sends a request to a service provider, there is normally an expectation that there will be a response, and that, following the sending of the request, the message handlers in the pipeline will be invoked again

when the response arrives. Some Web services do not send a response, and so you must take special action to indicate that CICS should not wait for a response before invoking the message handlers for a second time.

To do this, ensure that container DFHNORESPONSE is present at the end of pipeline processing in the request phase. Typically, this is done by application level code, because the knowledge of whether a response is expected is lodged in the application:

- For applications deployed with the CICS Web services assistant, CICS code will create the container.
- Applications that are not deployed with the assistant will typically create the container before invoking the application.

If you create or destroy container DFHNORESPONSE in a message handler, you must be sure that doing so will not disturb the message protocol between the service requester and the provider.

Working with messages in a terminal message handler

A typical terminal handler processes a request, invokes an application program, and generates a response.

Note: Although Web services typically use SOAP messages which contain XML, your message handlers will work as well with other message formats

In a terminal message handler, you can work with a request, and - optionally - generate a response and pass it back along the pipeline. A typical terminal handler will use the request as input to an application program, and use the application program's response to construct the response.

1. Using the contents of container DFHFUNCTION, determine that the message passed to this handler is a request, and that the handler is being called as a terminal handler.

DFHFUNCTION	Request or response	Type of handler	Inbound or outbound
PROCESS-REQUEST	Request	Terminal	Inbound

Tip:

- If DFHFUNCTION contains any other value, the handler is not a terminal handler, and these steps do not apply.
2. Retrieve the request from container DFHREQUEST. Container DFHRESPONSE is also present, with a length of zero.
 3. Perform any processing of the message which is required. Typically, a terminal handler will invoke an application program.
 4. Construct your response, and put it in container DFHRESPONSE. If there is no response, you must delete container DFHRESPONSE.

The response is passed to the next handler in the response phase of the pipeline. The handler is invoked for function SEND-RESPONSE. If there is no response, the next handler is invoked for function NO-RESPONSE.

Handling errors

Message handlers should be designed to handle errors that may occur in the pipeline.

When an error occurs in a message handler program, the program is invoked again for error processing. Error processing always takes place in the response phase of the pipeline; if the error occurred in the request phase, subsequent handlers in the request phase are bypassed.

In most cases, therefore, you should write your handler program to handle any errors that may occur.

1. Check that container DFHFUNTION contains HANDLER_ERROR, indicating that the message handler has been called for error processing.

Tip:

- If DFHFUNTION contains any other value, the message handler has not been invoked for error processing, and these steps do not apply.
2. Analyze the error information, and determine if the message handler can recover from the error by constructing a suitable response.
Container DFHERROR holds information about the error. For detailed information about this container, see “Container DFHERROR” on page 159. Container DFHRESPONSE is also present, with a length of zero.
 3. Perform any recovery processing.
 - If the message handler can recover, construct a response, and return it in container DFHRESPONSE.
 - If the message handler can recover, but no response is required, delete container DFHRESPONSE, and return container DFHNORESPONSE instead.
 - If the message handler cannot recover, return container DFHRESPONSE unchanged (that is, with a length of zero).

If your message handler is able to recover from the error, pipeline processing continues normally. If not, CICS generates a SOAP fault that contains information about the error. In the case of a transaction abend, the abend code is included in the fault.

The message handler interface

The CICS pipeline links to the message handlers using a channel containing a number of containers. Some containers are optional, others are required by all message handlers, and others are used by some message handlers, and not by others.

Before a handler is invoked, some or all of the containers are populated with information which the handler can use to perform its work. The containers returned by the handler determine the subsequent processing, and are passed on to later handlers in the pipeline.

Chapter 11. The SOAP message handlers

The SOAP message handlers are CICS-provided message handlers that you can include in your pipeline to process SOAP 1.1 and SOAP 1.2 messages. You can use the SOAP message handlers in a service requester or in a service provider pipeline.

On input, the SOAP message handlers parse inbound SOAP messages, and extract the SOAP <Body> element for use by your application program. On output, the handlers construct the complete SOAP message, using the <Body> element which your application provides.

If you use SOAP headers in your messages, the SOAP handlers can invoke user-written header processing programs that allow you to process the headers on inbound messages, and to add them to outbound messages.

SOAP message handlers, and any header processing programs, are specified in the pipeline configuration file, using the <cics_soap_1.1_handler> and the <cics_soap_1.2_handler> elements, and their sub-elements.

Typically, you will need just one SOAP handler in a pipeline. However, there are some situations where more than one is needed. For example, you can ensure that SOAP headers are processed in a particular sequence by defining multiple SOAP handlers.

Header processing programs

Header processing programs are user-written CICS programs that are linked to from the CICS-provided SOAP 1.1 and SOAP 1.2 message handlers, in order to process SOAP header blocks.

You can write your header processing program in any of the languages which CICS supports, and use any CICS command in the DPL subset. Your header processing program can link to other CICS programs.

The header processing programs have a channel interface; the containers hold information which the header program can examine or modify, including:

- The SOAP header block for which the program is invoked
- The SOAP message body

Other containers hold information about the environment in which the header program is invoked, such as:

- The transaction ID under which the header program was invoked
- Whether the program was invoked for a service provider or requester pipeline
- Whether the message being processed is a request or response

Header processing programs normally run under transaction CPIH, which is defined with the attribute TASKDATALOC(ANY). Therefore, when you link-edit the program, you must specify the AMODE(31) option.

How header processing programs are invoked for a SOAP request

The `<cics_soap_1.1_handler>` and `<cics_soap_1.2_handler>` elements in a pipeline configuration contain zero, one, or more, `<headerprogram>` elements, each of which contains the following children:

```
<program_name>
<namespace>
<localname>
<mandatory>
```

When a pipeline is processing an inbound SOAP message (a request in the case of a service provider, a response in the case of a service requester), the header program specified in the `<program_name>` element is invoked or not, depending upon:

- The contents of the `<namespace>`, `<localname>`, and `<mandatory>` elements
- The value of certain attributes of the root element of the SOAP header itself (the **actor** attribute for SOAP 1.1; the **role** attribute for SOAP 1.2)

The following rules determine if the header program will be invoked in a given case:

The `<mandatory>` element in the pipeline configuration file

If the element contains `true` (or `1`), the header processing program is invoked at least once, even if none of the headers in the SOAP message is selected for processing by the remaining rules:

- If none of the header blocks is selected, the header processing program is invoked once.
- If any of the header blocks is selected by the remaining rules, the header processing program is invoked once for each selected header.

Attributes in the SOAP header block

For SOAP 1.1, a header block is eligible for processing only if the **actor** attribute is absent, or has a value of `http://schemas.xmlsoap.org/soap/actor/next`

For SOAP 1.2, a header block is eligible for processing only if the **role** attribute is absent, or has one of the following values:

```
http://www.w3.org/2003/05/soap-envelope/role/next
http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver
```

A header block that is eligible for processing is not processed unless it is selected by the next rule.

The `<namespace>` and `<localname>` elements in the pipeline configuration file

A header block that is eligible for processing according to the previous rule is selected for processing only if:

- the name of the root element of the header block matches the `<localname>` element in the pipeline configuration file
- *and* the root element's namespace matches the `<namespace>` element in the pipeline configuration file

For example, consider this header block:

```
<t:myheaderblock xmlns:t="http://mynamespace" ...> .... </t:myheaderblock>
```

Subject to the other rules, the header block will be selected for processing when the following is coded in the pipeline configuration file:

```
<namespace>http://mynamespace</namespace>  
<localname>myheaderblock</localname>
```

The `<localname>` can contain an `*` to indicate that all header blocks in the namespace should be processed. Therefore, the same header block will be selected by the following:

```
<namespace>http://mynamespace</namespace>  
<localname>*</localname>
```

When the SOAP message contains more than one header, the header processing program is invoked once for each matching header, but the sequence in which the headers are processed is undefined.

The CICS-provided SOAP message handlers select the header processing programs that will be invoked based upon the header blocks that are present in the SOAP message at the time when the message handler receives it. Therefore, a header processing program is never invoked as a result of a header block that is added to a message in the same SOAP message handler. If you want to process the new header (or any modified headers) in your pipeline, you must define another SOAP message handler in your pipeline.

For an outbound message (a request in a service requester, a response in a service provider) the CICS-provided SOAP message handlers create a SOAP message that does not contain any headers. In order to add one or more headers to the message, you must write a header handler program to add the headers. To ensure that this header handler is invoked, you must define it in your pipeline configuration file, and specify `<mandatory>>true</mandatory>`.

If a header handler is invoked in the request phase of a pipeline, it will be invoked again in the response phase, even if the message that flows in the response phase does not contain a matching header.

The header processing program interface

The CICS-provided SOAP 1.1 and SOAP 1.2 message handlers link to the header processing programs using channel DFHHHC-V1. The containers that are passed on the channel include several that are specific to the header processing program interface, and sets of *context containers* and *user containers* that are accessible to all the header processing programs and message handler programs in the pipeline.

Container DFHHEADER is specific to the header processing program interface. Other containers are available elsewhere in your pipeline, but have specific uses in a header processing program. The containers in this category are:

- DFHWS-XMLNS
- DFHWS-BODY

Container DFHHEADER

When the header processing program is invoked, DFHHEADER contains the single header block which caused the header processing program to be driven. When the header program is specified with `<mandatory>>true</mandatory>` or `<mandatory>1</mandatory>` in the pipeline configuration file, it is invoked even when there is no matching header block in the SOAP message. In this case, container DFHHEADER has a length of zero. This will be the case when a header processing program is invoked to add a header block to a SOAP message that does not have header blocks.

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>`.

When the header program returns, container DFHHEADER must contain zero, one, or more header blocks which CICS inserts in the SOAP message in place of the original:

- You can return the original header block unchanged.
- You can modify the contents of the header block.
- You can append one or more new header blocks to the original block.
- You can replace the original header block with one or more different blocks.
- You can delete the header block completely.

Container DFHWS-XMLNS

When the header processing program is invoked, DFHWS-XMLNS contains information about XML namespaces that are declared in the SOAP envelope. The header program can use this information:

- to resolve qualified names that it encounters in the header block
- to construct qualified names in new or modified header blocks.

The namespace information consists of a list of namespace declarations, which use the standard XML notation for declaring namespaces. The namespace declarations in DFHWS-XMLNS are separated by spaces. For example:

```
xmlns:na='http://abc.example.org/schema' xmlns:nx='http://xyz.example.org/schema'
```

You can add further namespace declarations to the SOAP envelope by appending them to the contents of DFHWS-XMLNS. However, namespaces whose scope is a SOAP header block or a SOAP body are best declared in the header block or the body respectively. You are advised not to delete namespace declarations from container DFHWS-XMLNS in a header processing program, because XML elements which are not visible in the program may rely on them.

Container DFHWS-BODY

Contains the body section of the SOAP envelope. The header processing program can modify the contents.

When the header processing program is invoked, DFHWS-BODY contains the SOAP `<Body>` element.

When the header program returns, container DFHWS-BODY must again contain a valid SOAP `<Body>`, which CICS inserts in the SOAP message in place of the original:

- You can return the original body unchanged.
- You can modify the contents of the body.

You must not delete the SOAP body completely, as every SOAP message must contain a `<Body>` element.

Context containers and user containers

As well as the containers described, the interface passes the *control containerscontrol containers, context containerscontext containers*, and “User containers” on page 170*user containers* on channel DFHHHC-V1.

For more information about these containers, see Chapter 12, “Containers used in the pipeline,” on page 159.

The SOAP handler interfaces

The SOAP handler has two interfaces with user-written programs: the header processing program interface, which passes information between the SOAP handler and a header processing program; and the application interface, which passes information between the SOAP handler and the target application.

The application interface

The application interface is a channel that is passed between a SOAP handler and the target application program when it is invoked with a channel interface. When the target is invoked with a COMMAREA interface, the channel is not available to the target application program..

The channel (named DFHAHC-V1) used by the application interface passes the following containers:

DFHWS-XMLNS

Contains a list of name-value pairs that map namespace prefixes to namespaces.

- On input, the list contains the namespaces that are in scope from the SOAP envelope.
- On output, the list contains the namespace data that is assumed to be in the envelope tag.

DFHWS-BODY

Contains the body section of the SOAP envelope. Typically, the application will modify the contents.

DFHNORESPONSE

In the request phase of a service requester pipeline, indicates that the service provider is not expected to return a response. The contents of container DFHNORESPONSE are undefined; message handlers that need to know if the service provider is expected to return a response need only determine if the container is present or not:

- If container DFHNORESPONSE is present, then no response is expected.
- If container DFHNORESPONSE is absent, then a response **is** expected.

The channel also passes all the context containers that were passed to the calling message handler. A header processing program may add containers to the channel; the added containers are passed as user containers to the next handler in the pipeline.

Dynamic routing of inbound requests in a terminal handler

When the terminal handler of a service provider pipeline is one of the CICS-supplied SOAP message handlers, the target application program specified in container DFHWS-APPHANDLER is, in some cases, eligible for dynamic routing.

The transaction that runs the target application program is eligible for routing when one of the following is true:

- The target program is defined as DYNAMIC. For applications deployed with the CICS Web services assistant, the target program is DFHPITP
- A program in the pipeline has changed the contents of container DFHWS-USERID from its initial value
- A program in the pipeline has changed the contents of container DFHWS-TRANID from its initial value.

In addition, the CICS region that you are routing to must support channels and containers.

The routing will only take place if the TRANSACTION definition for the transaction named in DFHWS-TRANID specifies one of the following sets of attributes:

DYNAMIC(YES)

The transaction is routed using the distributed routing model, in which the routing program is specified in the **DSRTPGM** system initialization parameter.

DYNAMIC(NO)

REMOTESYSTEM(sysid)

The transaction is routed to the system identified by *sysid*.

For more information, see Routing inbound Web service requeststhe *CICS Customization Guide*.

For applications deployed with the CICS Web services assistant, there is a second opportunity to dynamically route the request, at the point where CICS links to the user's program. At this point, the request is routed using the dynamic routing model, in which the routing program is specified in the **DTRPGM** system initialization parameter. Eligibility for routing is determined, in this case, by the characteristics of the program. If you are using a channel and containers when linking to the program, you can only dynamically route the request to CICS regions that are at 3.1 or higher. If you are using a COMMAREA, this restriction does not apply.

For more information, see Routing DPL requests dynamicallythe *CICS Customization Guide*.

Chapter 12. Containers used in the pipeline

A pipeline typically consists of a number of message handler programs and, when the CICS-supplied SOAP message handlers are used, a number of header processing programs. CICS uses containers to pass information to and from these programs. The programs also use containers to communicate with other programs in the pipeline.

The CICS pipeline links to the message handlers and to the header processing programs using a channel with a number of containers. Some containers are optional, others are required by all message handlers, and others are used by some message handlers, and not by others.

Before a handler is invoked, some or all of the containers are populated with information which the handler can use to perform its work. The containers returned by the handler determine the subsequent processing, and are passed on to later handlers in the pipeline.

The containers can be categorized as:

Control containers

These are essential to the operation of the pipeline. Handlers can use the control containers to modify the sequence in which the handlers are processed. The names of the control containers are defined by CICS, and begin with the characters DFH.

Context containers

These contain information about the environment in which the handlers are called. CICS puts information in these containers before it invokes the first message handler, but, in some cases, the handlers are free to change the contents, or delete the containers. Changes to the context containers do not directly affect the sequence in which the handlers are invoked. The names of the context containers are defined by CICS, and begin with the charactersDFH.

Header processing program containers

These contain information that is used by header processing programs that are invoked from the CICS-supplied SOAP message handlers.

User containers

These contain information which one message handler needs to pass to another. The use of user containers is entirely a matter for the message handlers. You can choose your own names for these containers, but you must not use names that start with DFH.

The control containers

The control containers are essential to the operation of the pipeline. Handlers can use the control containers to modify the sequence in which the handlers are processed.

Container DFHERROR

DFHERROR is a container of DATATYPE(BIT) that is used to convey information about pipeline errors to other message handlers.

Table 3. Structure of container DFHERROR. All fields in the structure contain character data.

Field name	Length (bytes)	Contents
PIISNEB-MAJOR-VERSION	1	"1"
PIISNEB-MINOR-VERSION	1	"1"
PIISNEB-ERROR-TYPE	1	A numeric value denoting the type of error. The values are described in Table 4.
PIISNEB-ERROR-MODE	1	<p>P The error occurred in a provider pipeline</p> <p>R The error occurred in a requester pipeline</p> <p>T The error occurred in a Trust client</p>
PIISNEB-ABCODE	4	The abend code when the error is associated with a transaction abend.
PIISNEB-ERROR-CONTAINER1	16	The name of the container when the error is associated with a container.
PIISNEB-ERROR-CONTAINER2	16	The name of the second container when the error is associated with more than one container.
PIISNEB-ERROR-NODE	8	The name of the handler program in which the error occurred.

Table 4. Values for field PIISNEB-ERROR-TYPE

Value of PIISNEB-ERROR-TYPE	Meaning
1	The handler program abended. The abend code is in field PIISNEB-ABCODE.
2	A container required by the handler was empty. The name of the container is in field PIISNEB-ERROR-CONTAINER1.
3	A container required by the handler was missing. The name of the container is in field PIISNEB-ERROR-CONTAINER1.
4	Two containers were passed to the handler when only one was expected. The names of the containers are in fields PIISNEB-ERROR-CONTAINER1 and PIISNEB-ERROR-CONTAINER2.
5	An attempt to link to the target program failed. If target program abended, the abend code is in container PIISNEB-ABCODE.
6	The pipeline manager failed to communicate with a remote server due to an error in the underlying transport.

Table 4. Values for field PIISNEB-ERROR-TYPE (continued)

Value of PIISNEB-ERROR-TYPE	Meaning
7	There is an error with the DFHWS-STSACTION container. It is missing, corrupt or contains an invalid value.
8	DFHPIRT failed to start the pipeline.
9	DFHPIRT failed to put a message in a container.
10	DFHPIRT failed to get a message from a container.
11	An unhandled error has occurred.

The COBOL declaration of the container's structure is this:

```
01 PIISNEB.
  02 PIISNEB-MAJOR-VERSION PIC X(1).
  02 PIISNEB-MINOR-VERSION PIC X(1).
  02 PIISNEB-ERROR-TYPE PIC X(1).
  02 PIISNEB-ERROR-MODE PIC X(1).
  02 PIISNEB-ABCODE PIC X(4).
  02 PIISNEB-ERROR-CONTAINER1 PIC X(16).
  02 PIISNEB-ERROR-CONTAINER2 PIC X(16).
  02 PIISNEB-ERROR-NODE PIC X(8).
```

The language copybooks that map the container are:

Table 5.

Language	Copybook
COBOL	DFHPIUCO
PL/I	DFHPIUCL
C and C++	dfhpiuch.h
Assembler	DFHPIUCD

Container DFHFUNCTION

DFHFUNCTION is a container of DATATYPE(CHAR) that contains a 16-character string that indicates where in a pipeline a program is being invoked.

The string has one of the following values. The rightmost character positions are padded with blank characters.

RECEIVE-REQUEST

The handler is a non-terminal handler in a service provider pipeline, and is being invoked to process an inbound request message. On entry to the handler, the message is in control container DFHREQUEST.

SEND-RESPONSE

The handler is a non-terminal handler in a service provider pipeline, and is being invoked to process an outbound response message. On entry to the handler, the message is in control container DFHRESPONSE.

SEND-REQUEST

The handler is being invoked by a pipeline that is sending a request; that is, in a service requester that is processing an outbound message

RECEIVE-RESPONSE

The handler is being invoked by a pipeline that is receiving a response; that is, in a service requester that is processing an inbound message

PROCESS-REQUEST

The handler is being invoked as the terminal handler of a service provider pipeline

NO-RESPONSE

The handler is being invoked after processing a request, when there is no response to be processed.

HANDLER-ERROR

The handler is being invoked because an error has been detected.

In a service provider pipeline that processes a request and returns a response, the values of DFHFUNTION that occur are RECEIVE-REQUEST, PROCESS-REQUEST, and SEND-RESPONSE. Figure 23 shows the sequence in which the handlers are invoked, and the values of DFHFUNTION that are passed to each handler.

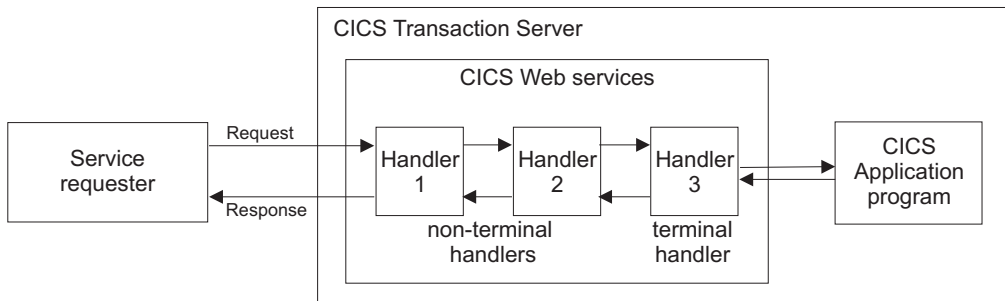


Figure 23. Sequence of handlers in a service provider pipeline

Sequence	Handler	DFHFUNTION
1	Handler 1	RECEIVE-REQUEST
2	Handler 2	RECEIVE-REQUEST
3	Handler 3	PROCESS-REQUEST
4	Handler 2	SEND-RESPONSE
5	Handler 1	SEND-RESPONSE

In a service requester pipeline, that sends a request and receives a response, the values of DFHFUNTION that occur are SEND-REQUEST and RECEIVE-RESPONSE. Figure 24 on page 163 shows the sequence in which the handlers are invoked, and the values of DFHFUNTION that are passed to each handler.

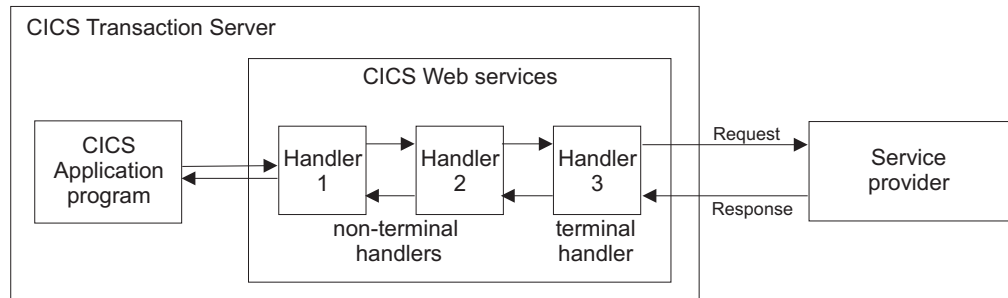


Figure 24. Sequence of handlers in a service requester pipeline

Sequence	Handler	DFHFUNTION
1	Handler 1	SEND-REQUEST
2	Handler 2	SEND-REQUEST
3	Handler 3	SEND-REQUEST
4	Handler 3	RECEIVE-RESPONSE
5	Handler 2	RECEIVE-RESPONSE
6	Handler 1	RECEIVE-RESPONSE

The values of DFHFUNTION that can be encountered in a given message handler depends upon whether the pipeline is a provider or requester, whether the pipeline is in the request or response phase, and whether the handler is a terminal handler or a non-terminal handler. The following table summarizes when each value can occur:

Value of DFHFUNTION	Provider or requester pipeline	Pipeline phase	Terminal or non-terminal handler
RECEIVE-REQUEST	Provider	Request phase	Non-terminal
SEND-RESPONSE	Provider	Response phase	Non-terminal
SEND-REQUEST	Requester	Request phase	Non-terminal
RECEIVE-RESPONSE	Requester	Response phase	Non-terminal
PROCESS-REQUEST	Provider	Request phase	Terminal
NO-RESPONSE	Both	Response phase	Non-terminal
HANDLER-ERROR	Both	Both	Both

Container DFHHTTPSTATUS

DFHHTTPSTATUS is a container of DATATYPE(CHAR) that is used to specify the HTTP status code and status text for a message produced in the response phase of a service provider pipeline.

The content of the DFHHTTPSTATUS container must be the same as the initial status line of an HTTP response message, which has the following structure:

```
HTTP/1.1 nnn tttttttt
```

HTTP/1.1

The version and release of HTTP.

nnn The 3-digit decimal HTTP status code to return.

ttttttt

The human-readable status text associated with the status code nnn.

The following string is an example of the content:

```
HTTP/1.1 412 Precondition Failed
```

The DFHHTTPSTATUS container is ignored when the pipeline uses the WebSphere MQ transport.

Container DFHMEDIATYPE

DFHMEDIATYPE is a container of DATATYPE(CHAR) that is used to specify the media type for a message produced in the response phase of a service provider pipeline.

The content of the DFHMEDIATYPE container must consist of a type and a subtype separated by a slash character. The following strings show two examples of correct content for the DFHMEDIATYPE container:

```
text/plain  
image/svg+xml
```

The DFHMEDIATYPE container is ignored when the pipeline uses the WebSphere MQ transport.

Container DFHNORESPONSE

DFHNORESPONSE is a container of DATATYPE(CHAR) that, in the request phase of a service requester pipeline, indicates that the service provider is not expected to return a response.

The contents of container DFHNORESPONSE are undefined; message handlers that need to know if the service provider is expected to return a response need only determine if the container is present or not:

- If container DFHNORESPONSE is present, then no response is expected.
- If container DFHNORESPONSE is absent, then a response **is** expected.

This information is provided, initially, by the service requester application, based upon the protocol used with the service provider. Therefore, it is inadvisable to delete this container in a message handler (or to create it, if it does not exist), as doing so may disturb the protocol between the end points.

Other than in the request phase of a service requester pipeline, the use of this container is not defined.

Container DFHREQUEST

DFHREQUEST is a container of DATATYPE(CHAR) that contains the request message that is processed in the request phase of a pipeline. If the message was constructed by a CICS-supplied SOAP message handler, and has not been changed subsequently, DFHREQUEST contains a complete SOAP envelope and all its contents in UTF-8 code page.

Container DFHREQUEST is present in the request when a message handler is invoked, and container DFHFUNTION contains RECEIVE-REQUEST or SEND-REQUEST.

In this situation, the normal protocol is to return DFHREQUEST to the pipeline with the same or modified contents. Processing of the pipeline's request phase continues normally, with the next message handler program in the pipeline (if there is one).

As an alternative, your message handler can delete container DFHREQUEST, and put a response in container DFHRESPONSE . If you do this, the normal sequence of processing is reversed, and the processing continues with the response phase of the pipeline.

Container DFHRESPONSE

DFHRESPONSE is a container of DATATYPE(CHAR) that contains the response message that is processed in the response phase of a pipeline. If the message was constructed by a CICS-supplied SOAP message handler, and has not been changed subsequently, DFHRESPONSE contains a complete SOAP envelope and all its contents in UTF-8 code page.

Container DFHRESPONSE is present when a message handler is invoked, and container DFHFUNTION contains SEND-RESPONSE or RECEIVE-RESPONSE.

In this situation, the normal protocol is to return DFHRESPONSE to the pipeline with the same or modified contents. Pipeline processing continues normally, with the next message handler program in the pipeline (if there is one).

Container DFHRESPONSE is also present (with a length of zero) when DFHFUNTION contains RECEIVE-REQUEST, SEND-REQUEST, PROCESS-REQUEST, or HANDLER-ERROR.

How containers control the pipeline protocols

The contents of the DFHFUNTION, DFHREQUEST, and DFHRESPONSE containers together control the pipeline protocols.

During the two phases of a pipeline's execution (the request phase and the response phase) the value of DFHFUNTION determines which control containers are passed to each message handler:

DFHFUNTION	Context	DFHREQUEST	DFHRESPONSE
RECEIVE-REQUEST	Service provider; request phase	Present (length > 0)	Present (length = 0)
SEND-RESPONSE	Service provider; response phase	Absent	Present (length > 0)
SEND-REQUEST	Service requester; request phase	Present (length > 0)	Present (length = 0)
RECEIVE-RESPONSE	Service requester; response phase	Absent	Present (length > 0)
PROCESS-REQUEST	Service provider; terminal handler	Present (length > 0)	Present (length = 0)
HANDLER-ERROR	Service requester or provider; either phase	Absent	Present (length = 0)
NO-RESPONSE	Service requester or provider; response phase	Absent	Absent

Subsequent processing is determined by which containers your message handler passes back to the pipeline:

During the request phase

- Your message handler can return container DFHREQUEST. Processing continues in the request phase with the next handler. The length of the data in the container must not be zero.
- Your message handler can return container DFHRESPONSE. Processing switches to the response phase, and the same handler is invoked with DFHFUNTION set to SEND-RESPONSE in a service provider, and RECEIVE-RESPONSE in a service requester. The length of the data in the container must not be zero.
- Your message handler can return no containers. Processing switches to the response phase, and the same handler is invoked with DFHFUNTION set to NO-RESPONSE.

In the terminal handler (service provider only)

- Your message handler can return container DFHRESPONSE. Processing switches to the response phase, and the previous handler is invoked with a new value of DFHFUNTION (SEND-RESPONSE). The length of the data in the container must not be zero.
- Your message handler can return no containers. Processing switches to the response phase, and the previous handler is invoked with a new value of DFHFUNTION (NO-RESPONSE).

During the response phase

- Your message handler can return container DFHRESPONSE. Processing continues in the response phase, and next handler is invoked. The length of the data in the container must not be zero.
- Your message handler can return no containers. Processing continues in the response phase, and the next handler in sequence is invoked with a new value of DFHFUNTION (NO-RESPONSE).

Important: During the request phase, your message handler can return DFHREQUEST or DFHRESPONSE, but not both. Since both containers are present when your message handler is invoked, you must delete one of them.

This table shows the action taken by the pipeline for all values of DFHFUNTION and all combinations of DFHREQUEST and DFHRESPONSE returned by each message handler.

DFHFUNCTION	Context	DFHREQUEST	DFHRESPONSE	Action	
RECEIVE-REQUEST	Service provider; request phase	Present (length > 0)	Present	(error)	
			Absent	Invoke the next handler with function RECEIVE-REQUEST	
		Absent	Present (length = 0)	Not applicable	(error)
			Present (length > 0)	Present (length > 0)	Switch to response phase, and invoke the same handler with function SEND-RESPONSE
			Present (length = 0)	Present (length = 0)	(error)
			Absent	Absent	Invoke the same handler with function NO-RESPONSE
SEND-RESPONSE	Service provider; response phase	Not applicable	Present (length > 0)	Invoke the previous handler with function SEND-RESPONSE	
			Present (length = 0)	(error)	
			Absent	Invoke the same handler with function NO-RESPONSE	
SEND-REQUEST	Service requester; request phase	Present (length > 0)	Present (length ≥ 0)	(error)	
			Absent	Invoke the next handler with function SEND-REQUEST	
		Absent	Present (length = 0)	Not applicable	(error)
			Present (length > 0)	Present (length > 0)	Switch to response phase, and invoke the previous handler with function RECEIVE-RESPONSE
			Present (length = 0)	Present (length = 0)	(error)
			Absent	Absent	Invoke the same handler with function NO-RESPONSE
RECEIVE-RESPONSE	Service requester; response phase	Not applicable	Present (length > 0)	Invoke the previous handler with function RECEIVE-RESPONSE	
			Present (length = 0)	(error)	
			Absent	Invoke the same handler with function NO-RESPONSE	
PROCESS-REQUEST	Service provider; terminal handler	Not applicable	Present (length > 0)	Invoke the previous handler with function RECEIVE-RESPONSE	
			Present (length = 0)	(error)	
			Absent	Invoke the same handler with function NO-RESPONSE	

DFHFUNCTION	Context	DFHREQUEST	DFHRESPONSE	Action
HANDLER-ERROR	Service requester or provider; either phase	Not applicable	Present (length > 0)	Invoke the previous handler with function SEND-RESPONSE or RECEIVE-RESPONSE
			Present (length = 0)	(error)
			Absent	Invoke the same handler with function NO-RESPONSE

The context containers

In some situations, user-written message handler programs, and header processing programs, need information about the context in which they are invoked. CICS provides this information in a set of *context containers* which are passed to the programs.

CICS initializes the contents of each context container, but, in some cases, you can change the contents in your message handler programs, and header processing program. For example, in a service provider pipeline in which the terminal handler is one of the CICS-provided SOAP handlers, you can change the userid and transaction ID of the target application program by modifying the contents of the appropriate context containers.

Some of the information provided in the containers applies only to a service provider, or only to a service requester, and therefore some of the context containers are not available in both.

Container DFH-HANDLERPLIST

DFH-HANDLERPLIST is a container of DATATYPE(CHAR) that is initialized with the contents of the appropriate <handler_parameter_list> element of the pipeline configuration file.

If you have not specified a handler parameter list in the pipeline configuration file, then the container is empty (that is, it has a length of zero).

You cannot change the contents of this container.

Container DFH-SERVICEPLIST

DFH-SERVICEPLIST is a container of DATATYPE(CHAR) that contains the contents of the <service_parameter_list> element of the pipeline configuration file.

If you have not specified a service parameter list in the pipeline configuration file, then the container is empty (that is, it has a length of zero).

You cannot change the contents of this container.

Container DFHWS-APPHANDLER

DFHWS-APPHANDLER is a container of DATATYPE(CHAR) that, in a service provider pipeline, is initialized with the contents of the <apphandler> element of the pipeline configuration file.

In the terminal handler of the pipeline, the CICS-supplied SOAP handlers get the name of the target application program from this container.

You can change the contents of this container in your message handlers or header processing programs.

CICS does not provide this container in a service requester pipeline.

Container DFHWS-DATA

DFHWS-DATA is a container of DATATYPE(BIT) that is used in service requester applications and optionally in service provider applications that are deployed with the CICS Web services assistant. It holds the top level data structure that is mapped to and from a SOAP request.

In service requester applications, container DFHWS-DATA must be present when a service requester program issues an EXEC CICS INVOKE WEBSERVICE command. When the command is issued, CICS converts the data structure that is in the container into a SOAP request. When the SOAP response is received, CICS converts it into another data structure that is returned to the application in the same container.

In service provider applications, container DFHWS-DATA is used by default when you do not specify the **CONTID** parameter on the DFHWS2LS or DFHWS2LS batch jobs. CICS converts the SOAP request message into the data structure that is passed to the application in the DFHWS-DATA container. The response is then saved in the same container, and CICS converts the data structure into a SOAP response message.

Container DFHWS-OPERATION

DFHWS-OPERATION is a container of DATATYPE(CHAR) that is normally used in a service provider application deployed with the CICS Web services assistant. It holds the name of the operation that is specified in a SOAP request.

In a service provider, the container supplies the name of the operation for which the application is being invoked. It is populated when a CICS-supplied SOAP message handler passes control to the target application program, and is visible only when the target program is invoked with a channel interface.

In a service requester pipeline, the container holds the name specified in the OPERATION option of the EXEC CICS INVOKE WEBSERVICE command. The container is not available to the application that issues the command.

Container DFHWS-PIPELINE

DFHWS-PIPELINE is a container of DATATYPE(CHAR) that contains the name of the PIPELINE in which the program is being run.

You cannot change the contents of this container.

Container DFHWS-SOAPLEVEL

DFHWS-SOAPLEVEL is a container of DATATYPE(BIT) that holds information about the level of SOAP used in the message that you are processing.

The container hold a binary fullword that indicates which level of SOAP is used for a Web service request or response:

- 1 The request or response is a SOAP 1.1 message.
- 2 The request or response is a SOAP 1.2 message.
- 10 The request or response is not a SOAP message.

You cannot change the contents of this container.

Container DFHWS-TRANID

DFHWS-TRANID is a container of DATATYPE(CHAR) that is initialized with the transaction ID of the task in which the pipeline is running.

If you change the contents of this container in a service provider pipeline in which the terminal handler is one of the CICS-supplied SOAP handlers (and you do so before control is passed to the target application program), the target application will execute in a new task with the new transaction ID.

Container DFHWS-URI

DFHWS-URI is a container of DATATYPE(CHAR) that, in a service provider, is initialized with the URI of the service. CICS extracts the URI from the incoming message.

Container DFHWS-USERID

DFHWS-USERID is a container of DATATYPE(CHAR) that is initialized with the user ID of the task in which the pipeline is running.

If you change the contents of this container in a service provider pipeline in which the terminal handler is one of the CICS-supplied SOAP handlers (and you do so before control is passed to the target application program), the target application will execute in a new task that is associated with the new userid. Unless you change the contents of container DFHWS-TRANID, the new task has the same transaction ID as the pipeline's task.

Container DFHWS-WEBSERVICE

DFHWS-WEBSERVICE is a container of DATATYPE(CHAR) that is used in a service provider pipeline only. It holds the name of the WEBSERVICE that specifies the execution environment when the target application has been deployed using the Web services assistant.

CICS does not provide this container in a service requester pipeline.

User containers

These contain information which one message handler needs to pass to another. The use of user containers is entirely a matter for the message handlers. You can choose your own names for these containers, but you must not use names that start with DFH.

Chapter 13. Support for Web Services transactions

The Web Services Atomic Transaction (or WS-AtomicTransaction) specification and the Web Services Coordination (or WS-Coordination) specification define protocols for short term transactions that enable transaction processing systems to interoperate in a Web services environment. Transactions that use WS-AtomicTransaction have the *ACID* properties of atomicity, consistency, isolation, and durability.

The specifications can be found at <http://www.ibm.com/developerworks/library/specification/ws-tx/>.

Note: CICS supports the August 2005 level of the specifications.

CICS applications that are deployed as Web service providers or requesters can participate in distributed transactions with other Web service implementations that support the specifications.

Registration services

Registration services is that part of the WS-Coordination model that enables an application to register for coordination protocols. In a distributed transaction, the registration services in the participating systems communicate with one another to enable the connected applications to participate in those protocols.

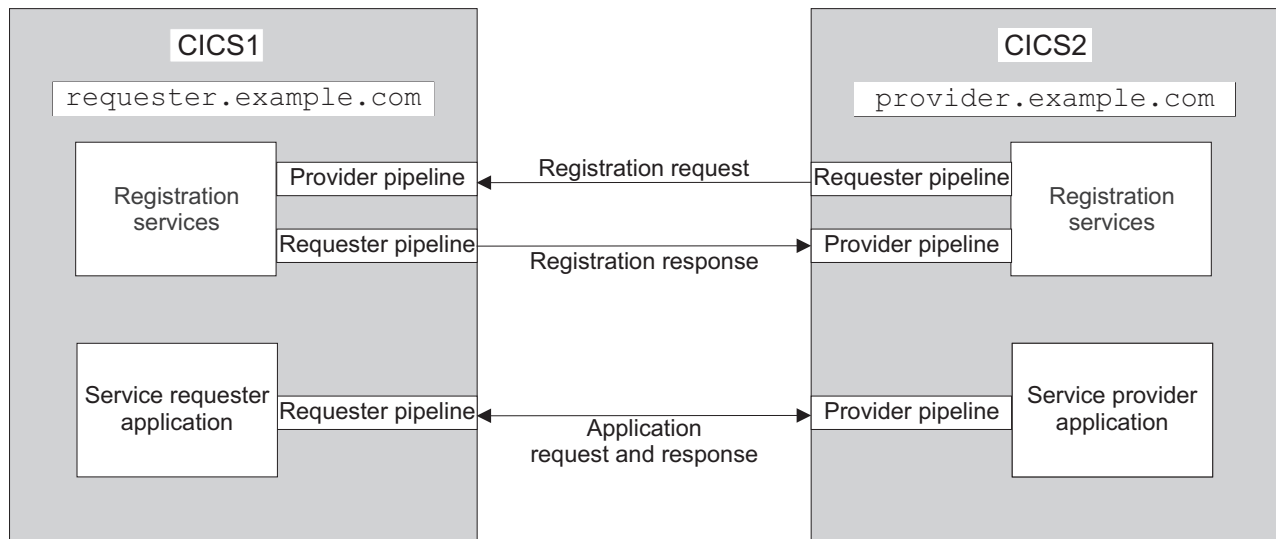


Figure 25. Registration services

Figure 25 shows two CICS systems, CICS1 and CICS2. A service requester application in CICS1 invokes a service provider application in CICS2. The two CICS regions and the applications are configured so that the two applications participate in a single distributed transaction, using the WS-Coordination protocols. The service requester application is the coordinator, and the service provider application is the participant.

In support of these protocols, the registration services in the two CICS regions interact at the start of the transaction, and again during transaction termination. During these interactions, registration services in both regions can operate at different times as a service provider and as a requester. Therefore, in each region, registration services use a service provider pipeline, and a service requester pipeline. The pipelines are defined to CICS with the PIPELINE and associated resources.

The registration services in each region are associated with an endpoint address. Thus, in the example, registration services in CICS1 has an endpoint address of requester.example.com; that in CICS2 has an endpoint address of provider.example.com.

In a CICSplex, you can distribute the registration services provider pipeline to a different region. This is shown in Figure 26.

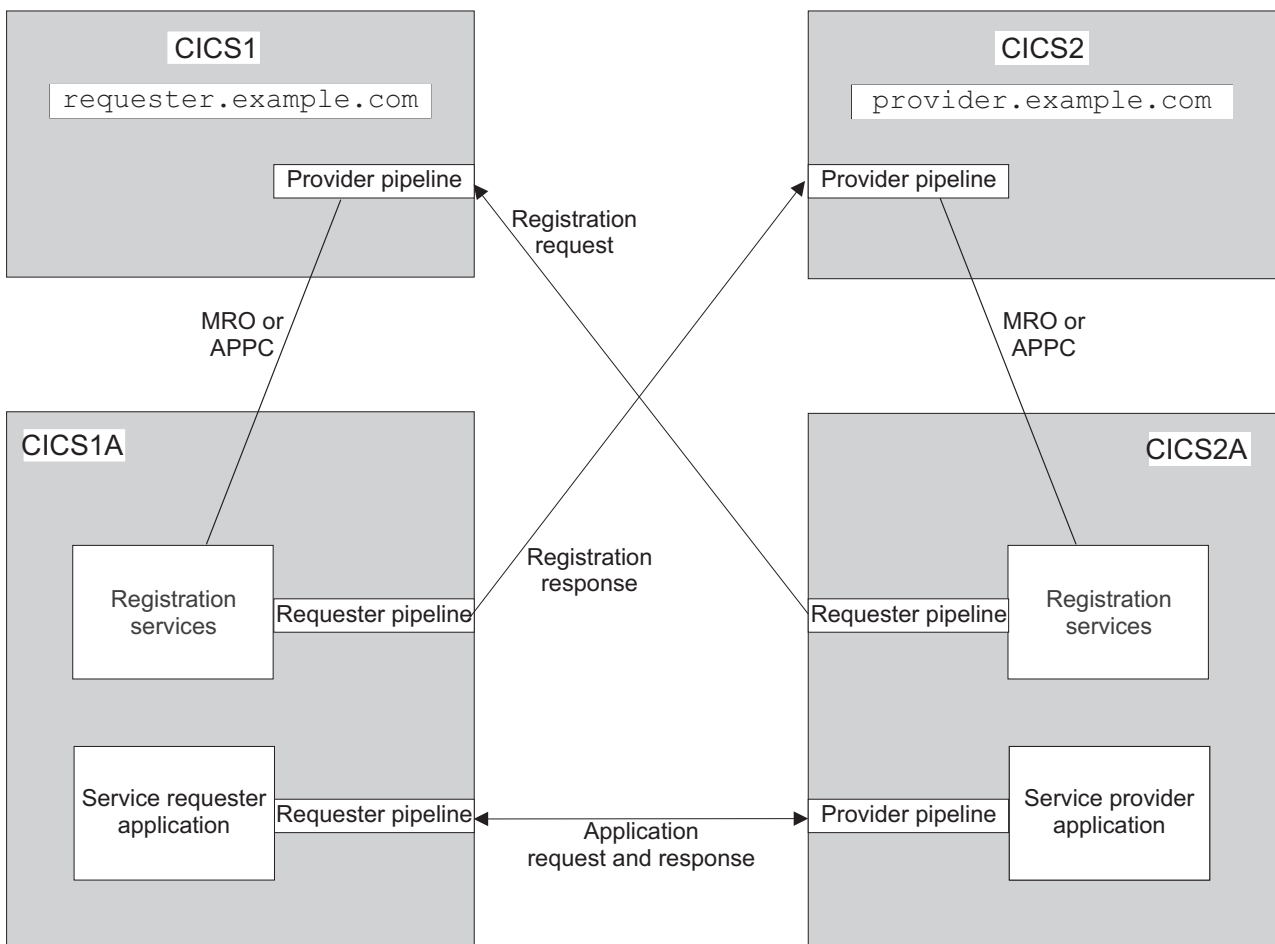


Figure 26. Registration services in a CICSplex®

In this configuration, the provider pipeline communicates with registration services using MRO or APPC. The registration services requester pipeline must remain in the same region as the application's requester pipeline.

This configuration is useful when your service requester and provider applications are distributed across a large number of regions. When you configure the application's pipelines to participate in Web service transactions, you must provide

information about the registration services endpoint by providing the IP address and port number of the registration services provider pipeline. By having a single endpoint, you can simplify configuration, because all your pipelines will contain the same information. For example, in Figure 26 on page 172 the IP address that you specify in the application's requester pipeline is `requester.example.com`.

The same arguments apply to the service provider application. In the example, the provider application's pipeline will specify an IP address of `requester.example.com`.

Configuring CICS for Web service transactions

For Web service requester and provider applications to participate in Web service transactions, you must configure CICS accordingly by installing a number of CICS resources.

Before you can install these resources you must know the location of the pipeline configuration files that CICS supplies in support of Web service transactions. By default, the configuration files are supplied in the `/usr/lpp/cicsts/cicsts31/pipeline/configs` directory, but the default file path might have been changed during CICS installation.

CICS support for Web service transactions uses a CICS-supplied registration services service provider and service requester, and you must install resources for both of these. Even if your applications are all service providers, or all service requesters, you must install both.

You must also install a program definition for the header handler program that is invoked when you run your service provider and requester applications.

The resources you require to configure CICS for Web service transactions are all supplied in the DFWSAT group, except for DFHPIDIR which is supplied in one of the following groups: DFHPIVS, DFHPIVR, or DFHPICF. The DFHWSAT group is not included in the DFHLIST list, and therefore is not installed automatically. You cannot change the resources supplied by CICS in the DFHWSAT group.

To configure CICS for Web service transactions:

1. Add the DFHPIDIR data set to your startup JCL. DFHPIDIR stores a mapping between contexts and tasks.
 - a. Add a new DD statement for the DFHPIDIR data set to your CICS startup JCL
 - b. Create the DFHPIDIR data set using information in DFHDEFDS.JCL. The default RECORDSIZE of DFHPIDIR is 1 KB, which is adequate for most uses. You can create DFHPIDIR with a larger RECORDSIZE if you need to.
 - c. Install the appropriate group for the data set on your CICS system: DFHPIVS, DFHPIVR, or DFHPICF.

If you want to share the DFHPIDIR file across CICS regions, the regions must be logically connected over MRO.

2. Copy the contents of the DFHWSAT group to another group. You cannot change the resources supplied by CICS in the DFHWSAT group. However, you must change the CONFIGFILE attribute in the PIPELINE resources.
3. Modify the CICS-supplied registration services provider PIPELINE resource. The PIPELINE is named DFHWSATP, and specifies pipeline configuration file `/usr/lpp/cicsts/cicsts31/pipeline/configs/registrationservicePROV.xml` in the CONFIGFILE attribute.

- a. Change the CONFIGFILE attribute to reflect the location of the file in your system.
- b. Leave the other attributes unchanged.

Use the pipeline configuration file exactly as provided; do not change its contents.

4. Install the PIPELINE resource. The registration services provider PIPELINE resource need not be in the same CICS region as your service requester or provider applications, but must be connected to that region with a suitable MRO or APPC connection.
5. Without changing it, install the URIMAP that is used by the registration services provider in the same region as the PIPELINE. The URIMAP is named DFHRSURI.
6. Modify the CICS-supplied registration services requester PIPELINE resource. The PIPELINE is named DFHWSATR, and specifies pipeline configuration file /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserviceREQ.xml in the CONFIGFILE attribute.
 - a. Change the CONFIGFILE attribute to reflect the location of the file in your system.
 - b. Leave the other attributes unchanged.

Use the pipeline configuration file exactly as provided; do not change its contents.

7. Install the PIPELINE resource. The registration services requester PIPELINE resource must be in the same CICS region as the service requester and provider applications.
8. Install the programs used by the registration service provider pipeline in the same region as your PIPELINE resources. The programs are DFHWSATX, DFHWSATR, and DFHPIRS. If both your PIPELINE resources are in different regions, you must install these programs in both regions.
9. Install the PROGRAM resource definition for the header handler program. The program is named DFHWSATH. Install the PROGRAM in the regions where your service provider and requester applications run.

CICS is now configured so that your service provider and requester applications can participate in distributed transactions using WS-AtomicTransaction and WS-Coordination protocols.

You must now configure each participating application individually.

Configuring a service provider for Web service transactions

If a service provider application is to participate in Web service transactions, the pipeline configuration file must specify a `<headerprogram>` and a `<service_parameter_list>`.

So that your service provider application can participate in Web service transactions, it must use SOAP protocols to communicate with the service requester, and you must configure your pipeline to use one of the CICS-provided SOAP message handlers. Even if you have configured your service provider application correctly, it will participate in Web service transactions with the service requester only if the requester application has been set up to participate.

In addition to the pipeline configuration information that is specific to your application, the configuration file must contain information that CICS uses to ensure that your application participates in Web service transactions.

CICS provides an example of a pipeline configuration file containing this information in file `/usr/lpp/cicsts/cicsts31/samples/pipelines/wsatprovider.xml`.

To configure a service provider for Web service transactions:

1. In the definition of your terminal handler, code a `<headerprogram>` element in the `<cics_soap_1.1_handler>` or `<cics_soap_1.2_handler>` element. Code the `<program_name>`, `<namespace>`, `<localname>`, and `<mandatory>` elements exactly as shown in this example:

```
<terminal_handler>
  <cics_soap_1.1_handler>
    <headerprogram>
      <program_name>DFHWSATH</program_name>
      <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoord</namespace>
      <localname>CoordinationContext</localname>
      <mandatory>>false</mandatory>
    </headerprogram>
  </cics_soap_1.1_handler>
</terminal_handler>
```

Include other `<headerprogram>` elements if your application needs them.

2. Code a `<registration_service_endpoint>` element in a `<service_parameter_list>`. Code the `<registration_service_endpoint>` as follows:

```
<registration_service_endpoint>
http://address:port/cicswsat/RegistrationService
</registration_service_endpoint>
```

where

address is the IP address of the CICS region where the registration service provider pipeline is installed.

port is the port number used by the registration service provider pipeline.

Code everything else exactly as shown; the string `cicswsat/RegistrationService` matches the PATH attribute of URIMAP DFHRSURI:

```
<registration_service_endpoint>
provider.example.com:7160/cicswsat/RegistrationService
</registration_service_endpoint>
```

Configuring a service requester for Web service transactions

If a service requester application is to participate in Web service transactions, the pipeline configuration file must specify a `<headerprogram>` and a `<service_parameter_list>`.

In order that your service requester application can participate in Web service transactions, it must use SOAP protocols to communicate with the service provider, and your pipeline must be configured to use one of the CICS-provided SOAP message handlers. Even if you have configured your service requester application correctly, it will only participate in Web service transactions with the service provider if the provider application has been set up to participate.

In addition to the pipeline configuration information that is specific to your application, the configuration file must contain information which CICS uses to ensure that your application participates in Web service transactions.

CICS provides an example of a pipeline configuration file containing this information in file `/usr/lpp/cicsts/cicsts31/samples/pipelines/wsatrequester.xml`.

1. Code a `<headerprogram>` element in the `<cics_soap_1.1_handler>` or `<cics_soap_1.2_handler>` element. Code the `<program_name>`, `<namespace>`, `<localname>`, and `<mandatory>` elements exactly as shown in the example below. For example:

```
<cics_soap_1.1_handler>
  <headerprogram>
    <program_name>DFHWSATH</program_name>
    <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoord</namespace>
    <localname>CoordinationContext</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.1_handler>
```

You can include other `<headerprogram>` elements if your application needs them.

2. Code a `<registration_service_endpoint>` element in a `<service_parameter_list>`. Code the `<registration_service_endpoint>` as follows:

```
<registration_service_endpoint>
http://address:port/cicswsat/RegistrationService
</registration_service_endpoint>
```

where

address is the IP address of the CICS region where the registration service provider pipeline is installed.

port is the port number used by the registration service provider pipeline.

There must be no space between the start the `<registration_service_endpoint>` element, its contents, and the end of the `<registration_service_endpoint>` element. Spaces have been included in this example for clarity.

3. If you want CICS to create a new transactional context for each request, rather than using the same one for requests in the same unit of work, add the empty element, `<new_tx_context_required/>`, in a `<service_parameter_list>` to your pipeline configuration file:

```
<service_parameter_list>
  <registration_service_endpoint>
    http://requester.example.com:7159/cicswsat/RegistrationService
  </registration_service_endpoint>
  <new_tx_context_required/>
</service_parameter_list>
```

There must be no space between the start the `<registration_service_endpoint>` element, its contents, and the end of the `<registration_service_endpoint>` element. Spaces have been included in this example for clarity.

The `<new_tx_context_required/>` setting is not the default for CICS, and is not included in the example pipeline configuration file, `wsatprovider.xml`. If you add `<new_tx_context_required/>` in a `<service_parameter_list>` to your pipeline configuration file, loopback calls to CICS are allowed, so be aware that a deadlock might occur in this situation.

Determining if the SOAP message is part of an atomic transaction

When a CICS Web service is invoked in the atomic transaction pipeline, the SOAP message does not necessarily have to be part of an atomic transaction.

The <soapenv:Header> element contains specific information when the SOAP message is part of an atomic transaction. To find out if the SOAP message is part of an atomic transaction, you can either:

- Look inside the contents of the <soapenv:Header> element using a trace.
 1. Perform an auxiliary trace using component PI and set the tracing level to 2.
 2. Look for trace point PI 0A31, which contains the information for the request container. In particular, look for PIIS EVENT - REQUEST_CNT which appears just before the <wsa:Action> element.
- Use a user-written message handler program in the DFHWSATP pipeline to display the content of the DFHREQUEST container when it contains the data RECEIVE-REQUEST. If you opt for this approach, make sure that you define the message handler program in the pipeline configuration file.

The following example shows the information that you could see in the SOAP envelope header for an atomic transaction.

```
<soapenv:Header>
  <wscoor:CoordinationContext soapenv:mustUnderstand="1"> 1
    <wscoor:Expires>500</wscoor:Expires>
    <wscoor:Identifier>com.ibm.ws.wstx:
      0000010a2b5008c80000000200000019a75aab901a1758a4e40e2731c61192a10ad6e921
    </wscoor:Identifier>
    <wscoor:CoordinationType>http://schemas.xmlsoap.org/ws/2004/10/wsat</wscoor:CoordinationType> 2
    <wscoor:RegistrationService 3
      xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          http://clientIPaddress:clientPort/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
        </wsa:Address>
        <wsa:ReferenceProperties
          xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          <websphere-wsat:txID
            xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">com.ibm.ws.wstx:
              0000010a2b5008c80000000200000019a75aab901a1758a4e40e2731c61192a10ad6e921
            </websphere-wsat:txID>
          <websphere-wsat:instanceID
            xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">com.ibm.ws.wstx:
              0000010a2b5008c80000000200000019a75aab901a1758a4e40e2731c61192a10ad6e921
            </websphere-wsat:instanceID>
          </wsa:ReferenceProperties>
        </wscoor:RegistrationService>
      </wscoor:CoordinationContext>
    </soapenv:Header>
```

1. The CoordinationContext indicates that the SOAP message is intended to participate in an atomic transaction. It contains the necessary information for the Web service provider to be part of the coordination service, assuming that the provider is configured to recognize and process the header.
2. The CoordinationType indicates the version of the WS-AT specification that the coordination context complies with.
3. The coordination RegistrationService describes where the coordinators registration point is, and the information that the participating Web service must return to the coordinator when it attempts to register as a component of the atomic transaction.

Checking the progress of an atomic transaction

When a CICS Web service is invoked as part of an atomic transaction, the transaction passes through a number of states. These states indicate whether the transaction was successful or had to roll back.

If you need to access this information, you can either:

- Look inside the contents of the <wsa:Action> element using a trace.
 1. Perform an auxiliary trace using component PI and set the tracing level to 2.
 2. Look for trace point PI 0A31, which contains the information for the request container. In particular, look for PIIS EVENT - REQUEST_CNT which appears just before the <wsa:Action> element.
- Use a user-written message handler program in the DFHWSATR and DFHWSATP pipelines to display the content of DFHWS-SOAPACTION containers. If you opt for this approach, make sure that you define the message handler program in the pipeline configuration files.

The states for a transaction that completes successfully and is committed are:

```
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register"  
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse"  
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare"  
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared"  
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit"  
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed "
```

The states for a transaction that is rolled back are:

```
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register"  
"http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse"  
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Rollback"  
"http://schemas.xmlsoap.org/ws/2004/10/wsat/Aborted"
```

Chapter 14. Support for Web Services Security

The *Web Services Security (WSS): SOAP Message Security 1.0* specification describes the use of *security tokens* and *digital signatures* to protect and authenticate SOAP messages.

Web Services Security protects the *privacy* and *integrity* of SOAP messages by, respectively, protecting messages from unauthorized disclosure and preventing unauthorized and undetected modification. WSS provides this protection by digitally signing and encrypting XML elements in the message. The elements that can be protected are the body, or any elements within the body or the header. Different levels of protection can be given to different elements within the SOAP message.

CICS Transaction Server for z/OS provides support for WSS: SOAP Message Security through the use of a CICS-supplied message handler, DFHWSSE1.

- For outbound messages, CICS provides support for digital signing and encryption of the entire SOAP body.
- For inbound messages, CICS supports messages in which the body, or elements of the body and header are encrypted or digitally signed.

CICS does not support Web Services Security for atomic transactions (WS-AT).

There is a significant performance impact when you use WSS to secure your Web services. The main advantage of implementing WSS is that by encrypting part of a SOAP message, you can send the message through a chain of intermediate nodes, all of which might have legitimate reasons to look at the SOAP header to make routing or processing decisions, but are not allowed to view the content of the message. By encrypting those sections that need to be confidential you:

- do not incur the overhead of encrypting and decrypting at every node in a chain of intermediate processes
- can route a confidential message over a public network of untrusted nodes, where only the ultimate recipient of the data can understand it.

If you want to use your own security procedures and processing, you can write a custom message handler to process secure SOAP messages in the pipeline. Read technote 1239021 on the IBM support site at <http://www.ibm.com/software/htp/cics/support/> for details of what you should include. For general information on how to write a custom message handler, see the *Application Development for CICS Web Services* redbook.

As an alternative to using Web Services Security, you can use SSL to encrypt the whole data stream.

Prerequisites

To implement Web Services Security, you must apply the following updates to your CICS region.

1. Install the free IBM XML Toolkit for z/OS V1.9, program number 5655-J51. You can download the toolkit from the following site: <http://www.ibm.com/servers/eserver/zseries/software/xml/>.
 - You must install version 1.9. Later versions do not work with Web Services Security support in CICS.

- The toolkit is an MVS feature and should be installed in the SMP/E zone for MVS.
 - Specify a valid keyring on the **KEYRING** system initialization parameter.
 - Apply the PTFs for APARs PK65352 and PK97657 to CICS, which change the required version of the toolkit from V1.7 to V1.9.
2. Apply ICSF APAR OA14956 if it is not already installed in your CICS region.
 3. Apply the PTF for APAR PK22736.
 4. Add the following libraries to the DFHRPL concatenation:
 - *hlq*.SIXMLOD1
 - *hlq*.SCEERUN
 - *hlq*.SDFHWSLD

where *hlq* is the high level qualifier that was specified by the system programmer when the Web Services Security APAR was installed.

The first three libraries contain DLLs that are required at run time by DFHWSSE1. IXM4C54 is provided by the XML toolkit and is found in *hlq*.SIXMLOD1; C128N is provided by the Language Environment runtime and is found in *hlq*.SCEERUN.

The *hlq*.SDFHWSLD library enables CICS to find the DFHWSSE1 and DFHWSXXX Web Services Security modules.

5. You might need to increase the value of the **EDSALIM** system initialization parameter. The three DLLs that need to be loaded require approximately 15MB of EDSA storage.

If you do not have the libraries specified, you get the following message:

```
CEE3501S The module module_name was not found.
```

The *module_name* varies depending on which library is missing.

The options for securing SOAP messages

CICS supports both signing and encrypting SOAP messages, so you can select the level of security that is most appropriate for the data that you are sending or receiving in the SOAP message.

The options that you can choose from are:

Basic authentication

In service provider mode, CICS can accept a *username token* in the SOAP message header for authentication on inbound SOAP messages. This is a type of security token that is comprised of a user name and password. CICS verifies the user name token using an external security manager such as RACF. If successful, the user name is placed in container DFHWS-USERID and the SOAP message is processed in the pipeline. If CICS is unable to verify the username token, a SOAP fault message is returned to the service requester.

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

Signing with X.509 certificates

In service provider and service requester mode, you can provide an X.509 certificate in the SOAP message header to sign the body of the SOAP message for authentication. This is a type of security token that is known as a *binary security token*. To accept binary security tokens from inbound

SOAP messages, the public key associated with the certificate must be imported into an external security manager, such as RACF, and associated with the key ring that is specified in the **KEYRING** system initialization parameter. 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 public keys.

When you specify the label associated with an X.509 digital certificate, do not use the following characters:

< > : ! =

You can also include a second X.509 certificate in the header and sign it using the first certificate. This allows you to run the work in CICS under the user ID associated with the second X.509 certificate. The certificate that you are using to sign the SOAP message must be associated with a trusted user ID, and have surrogate authority in order to assert that work should run under a different identity, the *asserted identity*, without the trusted user ID having the password associated with that identity.

Encrypting

In service provider and service requester mode, you can encrypt the SOAP message body using a symmetric algorithm such as Triple DES or AES. A symmetric algorithm is where the same key is used to encrypt and decrypt the data. This key is known as *asymmetric key*. It is then included in the message and encrypted using a combination of the intended recipient's public key and the asymmetric key encryption algorithm RSA 1.5. This provides you with increased security, because the asymmetric algorithm is complex and it is difficult to decrypt the symmetric key. However, you get better performance because the majority of the SOAP message is encrypted with the symmetric algorithm which is faster to decrypt.

For inbound SOAP messages, it is possible to encrypt an element in the SOAP body and then encrypt the SOAP body as a whole. This might be particularly appropriate for an element that contains sensitive data. If CICS receives a SOAP message with two levels of encryption, CICS decrypts both levels automatically. This is not supported for outbound SOAP messages.

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 mode, you can choose to both sign and encrypt a SOAP message. CICS always signs the SOAP message body first and then encrypts it. The advantage of this method is that it gives you both message confidentiality and integrity.

Signing of SOAP messages

For inbound messages, CICS supports digital signatures on elements in the SOAP body, and on SOAP header blocks. For outbound messages, CICS signs all elements in the SOAP body.

A SOAP message is an XML document, consisting of an <Envelope> element, which contains an optional <Header> element, and a mandatory <Body> element.

WSS: SOAP Message Security permits the contents of the <Header> and the <Body> to be signed at the element level. That is, in a given message, individual elements

can be signed or not, or can be signed with different signatures or using different algorithms. For example, in a SOAP message used in an online purchasing application, it would be appropriate to sign elements that confirm receipt of an order, as these may have legal status. However, to avoid the overhead of signing the entire message, other information might safely be left unsigned.

For inbound messages, message handler DFHWSSE1 can verify the digital signature on individual elements in the SOAP <Header> and the <Body>.

- DFHWSSE1 will verify signed elements it encounters in the <Header>.
- DFHWSSE1 will verify signed elements in the SOAP <Body>. If the handler is configured to expect a signed body, CICS will reject with a fault any SOAP message in which the body is not signed.

For outbound messages, message handler DFHWSSE1 can sign the SOAP <Body> only; it does not sign the <Header>. The algorithm and key used to sign the body are specified in the handler's configuration information.

Signature algorithms

CICS supports the signature algorithms required by the XML Signature specification. Each algorithm is identified by a universal resource identifier (URI).

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

Note that the DSA with SHA1 signature algorithm is supported on inbound SOAP messages only.

Example of a signed SOAP message

This is an example of a SOAP message that has been signed by CICS.

```
<?xml version="1.0" encoding="UTF8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
  <wsse:Security xmlns:ds="http://www.w3.org/2000/09/xmlsig#"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:xenc="http://www.w3.org/2001/04/xmenc#" SOAP-ENV:mustUnderstand="1">
    <wsse:BinarySecurityToken 1
      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#X509"
      wsu:Id="x509cert00">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQFADAwMQswCQYDVQQGEwJHcWJEMMAoGA1UEChMD
      SUJNMRMwEQYDVQQDEwpxaWxsIF1hdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
      MDELMAkGA1UEBhMCR0lXDDAKBgNVBAoTA01CTTETMBEGA1UEAxMKV21sbCBZYXR1czCBnzANBgkq
      hkiG9w0BAQEFAA0BjQAwYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
      6Ku4SuHbBwJtWNR0xBTAAS91Ea70yhVdppx0nJB0CiERg7S0HUdP7a8JXPfzA+BqV63JqRgJyxN6
      msfTAvEMR07LIXmZAte62nwcFrvCKNPFJIJ5mkaJ9v1p7jkCAwEAaA0BrTCBqjA/BglghkgBhvC
      AQQEMhMwR2VuZXJhdGVkIGJ5IHRoZSB0ZSB0ZSB0ZSB0ZSB0ZSB0ZSB0ZSB0ZSB0ZSB0ZSB0ZSB0
      ZQVRFU0BV5y5JQk0uQ09ggdJQk0uQ09NhgT XV1cuSUJNLkNPTYcECRR1BjAO
    </wsse:BinarySecurityToken>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
      <ds:SignedInfo xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ds="http://www.w3.org/2000/09/xmlsig#"
        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/xmenc#">
```

```

<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 "/>
</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 "/>
    </ds:Transform>
  </ds:Transforms>
  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/> 2
  <ds:DigestValue>QORZEA+gpaflUShspHxhrrjaF1XE=</ds:DigestValue> 3
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>drDH0XESiyN6YJm27mfK1ZMG4Q4IsZqQ9N9V6kEnw21k7aM3if77XNFnyKS4deg1bc3ga11kkaFJ 4
  p4jL0mYRqqycDPqPm+UEu7mzfHRQGe7H0EnFqZpikNqZK5FF6fvY1v2JgTDPwrOSYXmhzwegUDT
  1TVj0vuUgXYrFya03pw=</ds:SignatureValue>

<ds:KeyInfo>
  <wsse:SecurityTokenReference>
    <wsse:Reference URI="#x509cert00"
      ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/> 5
    </wsse:SecurityTokenReference>
  </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"
  <getVersion xmlns="http://msgsec.wssecfvt.ws.ibm.com"/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

1. The binary security token contains the base64Binary encoding of the X.509 certificate. This includes the public key that the intended recipient of the SOAP message should use to verify the signature.
2. The algorithm that is used during the hashing process to produce the message digest.
3. The value of the message digest.
4. The digest value is then encrypted with the user's private key and included here as the signature value.
5. References the binary security token that contains the public key that should be used to verify the signature.

CICS support for encrypted SOAP messages

For inbound messages, CICS can decrypt any encrypted elements in the SOAP body, and encrypted SOAP header blocks where the body is also encrypted. For outbound messages, CICS encrypts the entire SOAP body.

A SOAP message is an XML document, consisting of an <Envelope> element, which contains an optional <Header> element, and a mandatory <Body> element.

WSS: SOAP Message Security allows some of the contents of the <Header> and all of the contents of the <Body> to be encrypted at the element level. That is, in a given message, individual elements can have different levels of encryption, or can be encrypted using different algorithms. For example, in a SOAP message used in an online purchasing application, it would be appropriate to encrypt an individual's credit card details in order to ensure that they remain confidential. However, to avoid the overhead of encrypting the entire message, some information might safely be encrypted using a less secure (but faster) algorithm, and other information might safely be left unencrypted.

For inbound messages, the CICS-supplied message handler DFHWSSE1 can decrypt individual elements in the SOAP <Body>, and can decrypt elements in the SOAP <Header> if the SOAP body is also encrypted.

- DFHWSSE1 always decrypts elements it encounters in the <Header> in the order that the elements are found.
- DFHWSSE1 always decrypts elements in the SOAP <Body>. If you want to reject a SOAP message that does not have an encrypted <Body>, configure the handler to expect an encrypted body using the <expect_encrypted_body> element.

For outbound messages, message handler DFHWSSE1 supports encryption of the contents of the SOAP <Body> only; it does not encrypt any elements in the <Header>. When DFHWSSE1 encrypts the <Body>, all elements within the body are encrypted with the same algorithm and using the same key. The algorithm, and information about the key, are specified in the handler's configuration information.

Encryption algorithms

CICS supports the encryption algorithms required by the XML Encryption specification. Each algorithm is identified by a universal resource identifier (URI).

Algorithm	URI
Triple Data Encryption Standard algorithm (Triple DES)	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	http://www.w3.org/2001/04/xmlenc#aes128-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	http://www.w3.org/2001/04/xmlenc#aes192-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	http://www.w3.org/2001/04/xmlenc#aes256-cbc

Example of an encrypted SOAP message

This is an example of a SOAP message that has been encrypted by CICS.

```
<?xml version="1.0" encoding="UTF8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
  <wss:Security xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" SOAP-ENV:mustUnderstand="1">

    <wss: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#X509"
      wsu:Id="x509cert00">MIICDCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHcQYEMMAoGA1UEChMD
      SUJNMRMwEQYDVQQDEwXaWxsIF1hdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
      MDELMAMGA1UEBhMCR0IxDAAKBgNVBAoTAA0CTTETMBEGA1UEAxMKV21sbCBZYXRlczCBnZANBgkq
      hkiG9w0BAQEFAAOBjQAwYkCgYEArsRj/n+3RN75+jaxu0MBWShvZCB0egv8qu2UwLWEeiogePsR
      6Ku4SuHbBwJtWnr0xBTAAS91Ea70yhVdppx0nJB0CiERg7S0HUdP7a8JXPfzA+BqV63JqRgJyxN6
      msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvC
      AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBZBTZWN1cm10eSBTZXJ2ZXIgdGZm9yIHovT1MgKFJBQ0YpMDGg
      A1UdEQQxMC+BEVdZQVRFU0BV5y5JQk0uQ09NggdJQk0uQ09NghtXV1cuSUJNLKNTYcECRR1BjA0
      BgNVHQ8BAf8EBAMCAfYwHQYDVR00BBYEFMiPX6VZKP5+mSOY1TLNQGvVJzu+MAOGCSqGS1b3DQEB
      BQUAA4GBAHdrS409Jhoe67pHL2gs7x4SpV/NOuJnn/w25sjjop3RLGj2bKtK6RiEevhCDim6tnYW
      NyjBL1VdN7u5M6ktfd+HutR/HnIrQ3PkXZK4ipgC0RWDJ+8APLYSCxtFL+J0LN9Eo6yjIHL68mq
```

```

                                uZbTH2LvzFMy4PqEbmVKbmA87a1F
</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"/> 2
  <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"/> 3
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>M6bDQtJrvX0pEjAEIcf6bq6MP3ySmb4TQ0a/B5U1Qj1vWjD56V+GRJbF7ZCES5ojwCJHRVKW1ZB5 4
      Mb+aUzSw1soHzHQixc1JchgwCiyIn+E2TbG3R9m0zHD3XQsKTyVa0T1R7VPoMBd1ZLNDIomxjZn2
      p7JfxyWk0bcSLhdZnc=</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"/> 5
    <xenc:CipherData>
      <xenc:CipherValue>kgvqKnMcgIUn7r11vkFXF0g4SodEd3dxAJo/mVN6ef211B1MZe1g70yjEHf4ZXw1Cdt0FebId1nK 6
        rrksq11Mpw6So7ID8zav+KPQUKGm4+E</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

1. The binary security token contains the base64binary encoding of the X.509 certificate. This includes the public key that was used to encrypt the symmetric key.
2. States the algorithm that was used to encrypt the symmetric key.
3. References the binary security token that contains the public key used to encrypt the symmetric key.
4. The encrypted symmetric key that was used to encrypt the message.
5. The encryption algorithm that was used to encrypt the message.
6. The encrypted message.

Configuring RACF for Web Services Security

You must configure an external security manager, such as RACF, to create public-private key pairs and X.509 certificates for signing and encrypting outbound SOAP messages, and to authenticate and decrypt signed and encrypted inbound SOAP messages.

Before you perform this task, you must have RACF set up to work with CICS. The **DFLTUSER**, **KEYRING**, and **SEC=YES** system initialization parameters should be specified in the CICS region that contains your Web services pipelines.

1. To authenticate inbound SOAP messages that are signed:
 - a. Import the X.509 certificate into RACF as an ICSF key.
 - b. Attach the certificate to the key ring specified in the **KEYRING** system initialization parameter, using the RACDCERT command.

```

RACDCERT ID(userid1)
CONNECT(ID(userid2) LABEL('label-name') RING(ring-name)

```

where:

- *userid1* is the default user ID of the key ring or has authority to attach certificates to the key ring for other user IDs.
 - *userid2* is the user ID that you want to associate with the certificate
 - *label-name* is the name of the certificate
 - *ring-name* is the name of the key ring that is specified in the **KEYRING** system initialization parameter.
- c. Optional: If you want to use asserted identities, ensure that the user ID associated with the certificate has surrogate authority to allow work to run under other user IDs. You should also make sure that any additional certificates included in the SOAP message header are also imported into RACF.

The SOAP message can contain a binary security token in the header that either includes the certificate or contains a reference to the certificate. This reference can be the KEYNAME (this is the certificate label in RACF), a combination of the ISSUER and SERIAL number, or the SubjectKeyIdentifier. CICS can only recognize the SubjectKeyIdentifier if this has been specified as an attribute in the definition of the certificate in RACF.

2. To sign outbound SOAP messages:
 - a. Create an X.509 certificate and a public-private key pair using the following RACDCERT command.

```
RACDCERT ID(userid2) GENCERT
SUBJECTSDN(CN('common-name')
            T('title')
            OU('organizational-unit')
            O('organization')
            L('locality')
            SP('state-or-province')
            C('country'))
WITHLABEL('label-name')
```

where *userid2* is the user ID that you want to associate with the certificate. When you specify the certificate *label-name* value, do not use the following characters:

< > : ! =

- b. Attach the certificate to the key ring specified in the **KEYRING** system initialization parameter. Use the RACDCERT command.
- c. Export the certificate and publish it to the intended recipient of the SOAP message.

You can edit the pipeline configuration file so that CICS automatically includes the X.509 certificate in the binary security token of the SOAP message header for the intended recipient to validate the signature.

3. To decrypt inbound SOAP messages that are encrypted, the SOAP message must include the public key that is part of a key pair, where the private key is defined in CICS.
 - a. Generate a public-private key pair and certificate in RACF for encryption. The key pair and certificate should be generated using ICSF.
 - b. Attach the certificate to the key ring specified in the **KEYRING** system initialization parameter. Use the RACDCERT command.
 - c. Export the certificate and publish it to the generator of the SOAP messages that you want to decrypt.

The generator of the SOAP message can then import the certificate that contains the public key and use it to encrypt the SOAP message. The SOAP

message can contain a binary security token in the header that either includes the public key or contains a reference to it. This reference can be the KEYNAME, a combination of the ISSUER and SERIAL number, or the SubjectKeyIdentifier. CICS can only recognize the SubjectKeyIdentifier if this has been specified as an attribute in the definition of the public key in RACF.

4. To encrypt outbound SOAP messages:
 - a. Import the certificate that contains the public key that you want to use for encryption into RACF as an ICSF key. The intended recipient should have the private key associated with the public key to decrypt the SOAP message.
 - b. Attach the certificate that contains the public key to the key ring specified in the **KEYRING** system initialization parameter. Use the RACDCERT command.

CICS uses the public key in the certificate to encrypt the SOAP body, and sends the certificate containing the public key as a binary security token in the SOAP message header. This is defined in the pipeline configuration file.

Configuring CICS for Web Services Security

To configure CICS for Web Services Security (WSS), you must add a WSS handler to your pipeline configuration files.

Before performing this task, you must identify or create the pipeline configuration files to which you will add configuration information for WSS.

1. Add a `<wsse_handler>` element to your pipeline. The handler must be included in the `<service_handler_list>` element in a service provider or requester pipeline. Code the following elements:

```
<wsse_handler>
  <dfhwsse_configuration version="1">

  </dfhwsse_configuration>
</wsse_handler>
```

The `<dfhwsse_configuration>` element is a container for the other elements in the configuration file.

2. Optional: Code an `<authentication>` element.
 - In a service requester pipeline, the `<authentication>` element specifies that CICS should add an X.509 certificate to the security header in outbound SOAP messages.
 - 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.
 - a. Code the **trust** attribute to specify whether asserted identity is used, and the nature of the trust relationship between service provider and requester. For details of the **trust** attribute, see “The `<authentication>` element” on page 138.
 - b. Optional: If you specified **trust=none**, code the **mode** attribute to specify how credentials found in the message are processed. For details of the **mode** attribute, see “The `<authentication>` element” on page 138.
 - c. Within the `<authentication>` element, code the following:
 - 1) An optional `<certificate_label>` element that specifies the label associated with an X.509 digital certificate installed in RACF. If this element is specified in a service requester pipeline, and the `<suppress>`

element is not specified, the certificate is added to the security header in the SOAP message. If you do not specify a `<certificate_label>` element, CICS uses the default certificate in the RACF key ring. The certificate label should not contain any of the following characters:

`< > : ! =`

This element is ignored in a service provider pipeline.

- 2) An optional, empty `<suppress/>` element.

If this element is specified in a service provider pipeline, the handler will not attempt to use any security tokens in the message to determine under which user ID the work will run.

If this element is specified in a service requester pipeline, the handler will not attempt to add to the outbound SOAP message any of the security tokens that are required for authentication.

- 3) An `<algorithm>` element that specifies the URI of the algorithm used to sign the body of the SOAP message.

You can specify the following algorithms:

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

The DSA with SHA1 signature algorithm is supported on inbound SOAP messages only. If you are using basic authentication on inbound SOAP messages, you must still specify the `<algorithm>` element.

3. Optional: If you require inbound SOAP messages to be digitally signed, code an empty `<expect_signed_body/>` element.

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.

4. Optional: If you require inbound SOAP messages to be encrypted, code an empty `<expect_encrypted_body/>` element.

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.

5. Optional: If you require outbound SOAP messages to be signed, code a `<sign_body>` element.

- a. Within the `<sign_body>` element, code an `<algorithm>` element.
- b. Following the `<algorithm>` element, code a `<certificate_label>` element.

This is an example of a completed `<sign_body>` element:

```
<sign_body>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>SIGCERT01</certificate_label>
</sign_body>
```

6. Optional: If you require outbound SOAP messages to be encrypted, code an `<encrypt_body>` element.

- a. Within the `<encrypt_body>` element, code an `<algorithm>` element.
- b. Following the `<algorithm>` element, code a `<certificate_label>` element.

This is an example of a completed <encrypt_body> element:

```
<encrypt_body>
  <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
  <certificate_label>ENCCERT02</certificate_label>
</encrypt_body>
```

Example

The following example shows a completed <wsse_handler> in which all the optional elements are present:

```
<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>
```

Chapter 15. Diagnosing problems

The problems that you might get when implementing Web services in CICS can occur during the deployment process or at run time, when CICS is transforming SOAP messages.

Diagnosing deployment errors

Deployment errors can occur when you try to run the Web services assistant batch jobs or install a PIPELINE or WEBSERVICE resource in CICS. In the event of a deployment error, PIPELINE resources usually install in a DISABLED state, and WEBSERVICE resources install in an UNUSABLE state. The most common deployment errors are described here, including the symptom of the problem, the cause and the solution.

Information and error messages associated with the Web services assistant batch jobs are located in the job log. Error messages associated with installing resources are located in the system log.

- You receive a return code of 4, 8, or 12 when running the Web services assistant batch jobs DFHWS2LS or DFHLS2WS. The return codes mean the following:
 - 4 - Warning. The job completed successfully, but one or more warning messages have been issued.
 - 8 - Input error. The job did not complete successfully. One or more error messages were issued while validating the input parameters.
 - 12 - Error. The job did not complete successfully. One or more error messages were issued during execution.
 1. Check the job log for any warning or error messages. Look up the detailed explanations for the messages. The explanations normally describe actions that you can take to fix the problem.
 2. Ensure that you have entered the correct values for each of the parameters in the job. Parameter values such as file names and elements in the Web service description must be treated as case sensitive.
 3. Ensure that you have specified the correct combination of parameters. For example, if you include the **PGMNAME** parameter in DFHWS2LS when generating a Web service binding file for a service requester, you get an error and the job does not complete successfully.
- You receive a DFHPI0914 error message when attempting to install a WEBSERVICE resource. The message includes some information about the cause of the install failure.
 1. Check that you have authorized CICS to read the Web service binding file in z/OS UNIX.
 2. Check that the Web service binding file is not corrupt. This can occur, for example, if you use FTP to transfer the file to z/OS UNIX in text mode rather than binary mode.
 3. Check that two Web service binding files with the same name are not in different pick up directories.
 4. If you are attempting to install a resource for a Web service requester application, check that the version of the SOAP binding matches the level supported in the pipeline. You cannot install a SOAP 1.1 WEBSERVICE into a service requester pipeline that supports SOAP 1.2.

5. Check that you are not installing a provider mode WEBSERVICE resource into a requester mode pipeline. Provider mode Web service binding files specify a **PROGRAM** value, whereas requester mode binding files do not.
 6. If you are using DFHWS2LS or DFHLS2WS, check that you have specified the correct parameters when generating the Web service binding file. Some parameters, such as **PGMNAME**, are only allowed for Web service providers and have to be excluded if you are creating a Web service requester.
 7. If you are using DFHWS2LS or DFHLS2WS, check the messages issued by the job to see if there are any problems that you need to resolve before creating the WEBSERVICE resource.
- The PIPELINE resource fails to install and you receive a DFHPI0700, DFHPI0712, DFHPI0714 or similar error message.
 1. If you received a DFHPI0700 error message, you need to enable PL/I language support in your CICS region. This is required before you can install any PIPELINE resources. See Language Environment support for PL/I for more information.
 2. Check that you have authorized CICS to access the z/OS UNIX directories to read the pipeline configuration files.
 3. Check that the directory you are specifying in the **WSDIR** parameter is valid. In particular, check the case as directory and file names in z/OS UNIX are case-sensitive.
 4. Ensure that you do not have a PIPELINE resource of the same name in an ENABLED state in the CICS region.
 - The PIPELINE resource installs in a DISABLED state. You get an error message in the range of DFHPI0702 to DFHPI0711.
 1. Check that there are no errors in the pipeline configuration file. The elements in the pipeline configuration file can only appear in certain places. If you specify these incorrectly you get a DFHPI0702 error message. This message includes the name of the element that is causing the problem. Check the element description to make sure you have coded it in the correct place.
 2. Check that you do not have any unprintable characters, such as tabs, in the pipeline configuration file.
 3. Check that the XML is valid. If the XML is not valid, this can cause parsing errors when you attempt to install the PIPELINE resource.
 4. Ensure that the pipeline configuration file is encoded in US EBCDIC. If you try to use a different EBCDIC encoding, CICS cannot process the file.

Diagnosing service provider runtime errors

If you are having problems receiving or processing inbound messages in a provider mode pipeline, there could be a problem with the transport or a specific SOAP message.

- You receive a DFHPI0401, DFHPI0502 or similar message, indicating that a HTTP or WMQ transport error has occurred. If the transport is HTTP, the client receives a 500 Server Internal Error message. If the transport is WMQ, the message is written to the dead letter queue (DLQ). A SOAP fault is not returned to the Web service requester, because CICS is unable to determine what type of message was received.
 1. If you are using HTTP, check that you have specified the charset parameter on the Content-Type header of the Web service request; for example, charset=ISO-8859-1 During request processing of the inbound SOAP request,

CICS checks for a charset value and issues the DFHPI0401 message if this parameter is not present. The client receives a 500 Server Internal Error message.

- You receive a DFH-prefixed message, and a 404 Not Found error message.
 1. If you are not using the Web services assistant, you must create a URIMAP resource. If you are using the Web services assistant, the URIMAP is created automatically for you when you run the PIPELINE SCAN command. The system log provides information on any errors that occurred as a result of running this command.
 2. Check that the WEBSERVICE resource is enabled and that the URIMAP it is associated with is what you expected. If your WEBSERVICE resource has installed in an UNUSABLE state, see “Diagnosing deployment errors” on page 191.
 3. Check that you have correctly specified the URI and port number. In particular, check the case as the attribute PATH on the URIMAP resource is case sensitive.
- If there are unexpected errors being reported, consider using CEDX to debug the Web service application.
 1. Check the system log to see what error messages are being reported by CICS. This could give you an indication of what type of error is occurring. If CICS is not reporting any errors, ensure that the request is reaching CICS through the network.
 2. Run CEDX against CPIH for the HTTP transport, CPIQ for the WMQ transport, or the transaction that you specified in the URIMAP if this is different.

If a task switch occurs during the pipeline processing before the application handler, then unless the DFHWS-TRANID container is populated, the new task runs under the same transaction id as the first one. This can interfere with running CEDX, as the first task has a lock on the CEDX session. You can avoid this problem by using DFHWS-TRANID to change the transaction id when the task switches, allowing you to use CEDX on both the pipeline and application tasks separately.
 3. If CEDX doesn't activate or allow you solve the problem, consider running auxiliary trace with the PI, SO, AP, EI, and XS domains active. This could indicate whether there is a security problem, TCP/IP problem, application program problem or pipeline problem in your CICS region. Look for any exception trace points or abends.
- If you are receiving conversion errors, see “Diagnosing data conversion errors” on page 195.

Diagnosing service requester runtime errors

If you are having problems sending Web service requests from your service requester application, or you are receiving SOAP fault messages from the Web service provider, the problems could be due to errors in individual Web services or issues at the transport level.

- If you are using the INVOKE WEBSERVICE command in your application program, a RESP and RESP2 code are returned when there is a problem.
 1. Look up the meaning of the RESP and RESP2 codes for the INVOKE WEBSERVICE command to give you an indication of what the problem might be.
 2. Check the CICS system log to see if there are any messages that can help you determine the cause of the problem.

- If you are unable to send a SOAP request message and the pipeline is returning a DFHERROR container, there was a problem when the pipeline tried to process the SOAP message.
 1. Look at the contents of the DFHERROR container. This should contain an error message and some data describing the problem that occurred.
 2. Have you introduced any new message handlers or header processing programs in the pipeline? If you have, try removing the new program and rerunning the Web service to see if this solves the problem. If your message handler is trying to perform some processing using a container that isn't present in the pipeline, or is trying to update a container that is read-only, the pipeline stops processing and returns an error in the DFHERROR container. Header processing programs can only update a limited set of containers in the pipeline. See “The header processing program interface” on page 155 for details.
 3. If the Web service requester application is not using the INVOKE WEBSERVICE command to send a Web service request, check that it has created all of the necessary control containers and that they are the right datatype. In particular, check that the DFHREQUEST container has a datatype of CHAR rather than BIT.
 4. If the Web service requester application is using the INVOKE WEBSERVICE command an INVREQ and a RESP2 code of 14 is returned, this indicates that there has been a data conversion error. See “Diagnosing data conversion errors” on page 195.
 5. Check that the XML in your SOAP message has not been invalidated by a custom message handler during pipeline processing. CICS does not perform any validation on outbound messages in the pipeline. If your application uses the INVOKE WEBSERVICE command, the XML is generated by CICS and is well formed when the body of the SOAP message is placed in the DFHREQUEST container. However, if you have any additional message handlers that change the contents of the SOAP message, this is not validated in the pipeline.
- If you are able to send a SOAP message, but are getting a time out or transport error, this is normally returned as a SOAP fault.
 1. Check that the network end point is present.
 2. Ensure that the dispatcher timeout for the transaction meets your application's requirements. The DTIMEOUT attribute on the transaction defines how long CICS waits for a reply from the Web service provider before returning to the application.
- If you are able to send a SOAP message, but are getting a SOAP fault response back from the Web service provider that you didn't expect, look at the contents of the DFHWS-BODY container for details of the SOAP fault.
 1. If you sent a complete SOAP envelope in DFHREQUEST using the DFHPIRT interface, ensure that the outbound message doesn't contain duplicate SOAP headers. This can occur when the requester pipeline uses a SOAP 1.1 or SOAP 1.2 message handler. The SOAP message handlers add SOAP headers, even if they are already specified in the SOAP envelope by the service requester application. In this scenario, you can either:
 - Remove the SOAP 1.1 or SOAP 1.2 message handler from the pipeline. This will affect any other service requester applications that use this pipeline.
 - Remove the SOAP headers from the SOAP envelope that the application puts in DFHREQUEST. CICS adds the necessary SOAP headers for you. If you want to perform additional processing on the headers, you can use the header processing program interface.

- Use a WEB SEND command instead in your application and opt out of the Web services support.

Diagnosing data conversion errors

Data conversion errors can occur at run time when converting a SOAP message into a CICS COMMAREA or container and from a COMMAREA or container into a SOAP message.

Symptoms include the generation of SOAP fault messages and CICS messages indicating that a failure has occurred.

If you have a data conversion problem, you should perform the following steps:

1. Ensure that the WEBSERVICE resource is up to date. Regenerate the Web service binding file for the Web service and redeploy it to CICS.
2. Ensure that the remote Web service has been generated using the same version of the Web service document (WSDL) as used or generated by CICS.
3. If you are sure that the WEBSERVICE resource is using a current Web service binding file:
 - a. Enable runtime validation for the WEBSERVICE resource using the command `SET WEBSERVICE(name) VALIDATION` where *name* is the WEBSERVICE resource name.
 - b. Check for the CICS messages DFHPI1001 or DFHPI1002 in the message log. DFHPI1001 describes the precise nature of the data conversion problem, and should help you identify the source of the conversion error. DFHPI1002 indicates that no problems were found.
 - c. When you no longer need validation for the Web service, use the following command to turn validation off: `SET WEBSERVICE(name) NOVALIDATION`.
4. If you still have not determined the reason for the conversion error, take a CICS trace of the failing Web service. Look for the following PI domain exception trace entries:

```
PI 0F39 - PICC *EXC* - CONVERSION_ERROR
PI 0F08 - PIII *EXC* - CONVERSION_ERROR
```

A PICC conversion error indicates that a problem occurred when transforming a SOAP message received by CICS into a COMMAREA or container. A PIII conversion error indicates that a problem occurred when generating a SOAP message from a COMMAREA or container supplied by the application program. In both cases, the trace point identifies the name of the field associated with the conversion error and might also identify the value that is causing the problem. If either of these trace points appear, then it will be followed by a conversion error. For a possible interpretation of these conversion errors, see “Conversion errors in trace points” on page 196.

Why data conversion errors occur

CICS validates SOAP messages only to the extent that it is necessary to confirm that they contain well-formed XML, and to transform them. This means that it is possible for a SOAP message to be successfully validated using the WSDL, but then fail in the runtime environment and vice versa.

The WEBSERVICE resource encapsulates the mapping instructions to enable CICS to perform data conversion at run time. A conversion error occurs when the input does not match the expected data, as described in the WEBSERVICE resource.

This mismatch can occur for any of the following reasons:

- A SOAP message that is received by CICS is not well formed and valid when checked against the Web service description (WSDL) associated with the WEBSERVICE resource.
- A SOAP message that is received by CICS is well formed and valid but contains values that are out of range for the WEBSERVICE resource.
- The contents of a COMMAREA or container are not consistent with the WEBSERVICE resource and the language structure from which the Web service was generated.

For example, the WSDL document might specify range restrictions on a field, such as an unsignedInt that can only have a value between 10 and 20. If a SOAP message contains a value of 25, then validating the SOAP message would cause it to be rejected as invalid. The value 25 is accepted as a valid value for an integer and is passed to the application.

A second example is where the WSDL document specifies a string without specifying a maximum length. DFHWS2LS assumes a maximum length of 255 characters by default when generating the Web service binding file. If the SOAP message contains 300 characters, then although the check against the WSDL would validate the message as no maximum length is set, an error would be reported when attempting to transform the message as the value does not fit the 255 character buffer allocated by CICS.

Code page issues

CICS uses the value of the **LOCALCCSID** system initialization parameter to encode the application program data. However, the Web service binding file is encoded in US EBCDIC (Cp037). This can lead to problems with converting data when the code page used by the application program encodes characters differently to the US EBCDIC code page. To avoid this problem, you can use the **CCSID** parameter in the Web services assistant batch jobs to specify a different code page to encode data between the application program and the Web services binding file. The value of this parameter overrides the **LOCALCCSID** system initialization parameter for that particular WEBSERVICE resource. The *value* of **CCSID** must be an EBCDIC CCSID.

Conversion errors in trace points

When you run tracing for a failing Web service and find the PI domain exception trace points PI 0F39 or PI 0F08, a conversion error is provided by CICS. Possible interpretations for these conversion errors are provided to help you diagnose the cause of the conversion error, and where appropriate, next steps are also given.

The following conversion errors refer to COMMAREAs, but these errors can equally apply to containers.

INPUT_TOO_LONG

This conversion error occurs when:

- A SOAP element that is declared as numeric contains more than 31 digits
- A numeric field in the COMMAREA contains a value that is more than 31 digits in length.

OUTPUT_OVERFLOW

This conversion error occurs when:

- A SOAP element contains a value that is too long to fit in the associated field of the COMMAREA
- A SOAP element contains a numeric value that is outside the permitted range for the associated field in the COMMAREA.

Consider changing the Web service description (WSDL) to explicitly supply a “maxLength” facet for this field. If a “maxLength” is specified in the WSDL, CICS ensures that this much space is set aside in the COMMAREA for the field. If a “maxLength” facet is not specified, CICS uses a default of 255 characters. This might be an inappropriate value for the field.

You can also add a “whitespace” facet for character based fields and set it to “collapse”. This ensures that white space is removed from the field. By default, white space is preserved.

NEGATIVE_UNSIGNED

This conversion error occurs when:

- A negative number has been found in a SOAP element that is declared as unsigned.
- A negative number has been found in a COMMAREA field that is declared as unsigned.

NO_FRACTION_DIGITS

This conversion error occurs when a SOAP element contains a number that has a decimal point but is not followed by any valid fractional digits.

FRACTION_TOO_LONG

This conversion error occurs when a SOAP element contains a number with more nonzero fraction digits than the WSDL allows.

INVALID_CHARACTER

This conversion error occurs when:

- A SOAP element that is declared as a boolean contains a value other than 0, 1, true, or false.
- A SOAP element that is declared as hexBinary contains a value that is not in the range 0-9, a-f, A-F.
- A SOAP element that is declared as numeric contains a nonnumeric character
- A SOAP message is not well formed.

ODD_HEX_DIGITS

This conversion error occurs when a SOAP element that is declared as hexBinary contains an odd number of hexadecimal characters.

INVALID_PACKED_DEC

This conversion error occurs when a packed decimal field in the COMMAREA contains an illegal value that can not be converted to XML.

INVALID_ZONED_DEC

This conversion error occurs when a zoned decimal field in the COMMAREA contains an illegal value that can not be converted to XML.

INCOMPLETE_DBCS

This conversion error occurs when a DBCS sequence in the COMMAREA is missing a shift in (SI) character.

SOAP fault messages for conversion errors

If a conversion error occurs at run time and CICS is acting as a Web service provider, a SOAP fault message is issued to the service requester. This SOAP fault message includes the message that is issued by CICS.

The service requester can receive one of the following SOAP fault messages:

- Cannot convert SOAP message

This fault message implies that either the SOAP message is not well formed and valid, or its values are out of range.

- Outbound data cannot be converted

This fault message implies that the contents of a COMMAREA or container are not consistent.

- Operation not part of web service

This fault message is a special variation of when an invalid SOAP message is received by CICS.

If CICS is the Web service requester, the INVOKE WEBSERVICE command returns a RESP2 code of INVREQ and the value 14.

Chapter 16. The CICS catalog manager example application

The CICS catalog example application is a working COBOL application that is designed to illustrate best practice when connecting CICS applications to external clients and servers.

The example is constructed around a simple sales catalog and order processing application, in which the end user can perform these functions:

- List the items in a catalog.
- Inquire on individual items in the catalog.
- Order items from the catalog.

The catalog is implemented as a VSAM file.

The base application has a 3270 user interface, but the modular structure, with well-defined interfaces between the components, makes it possible to add further components. In particular, the application comes with Web service support, which is designed to illustrate how you can extend an existing application into the Web services environment.

The base application

The base application, with its 3270 user interface, provides functions with which you can list the contents of a stored catalog, select an item from the list, and enter a quantity to order. The application has a modular design which makes it simple to extend the application to support newer technology, such as Web services.

Figure 27 on page 200 shows the structure of the base application.

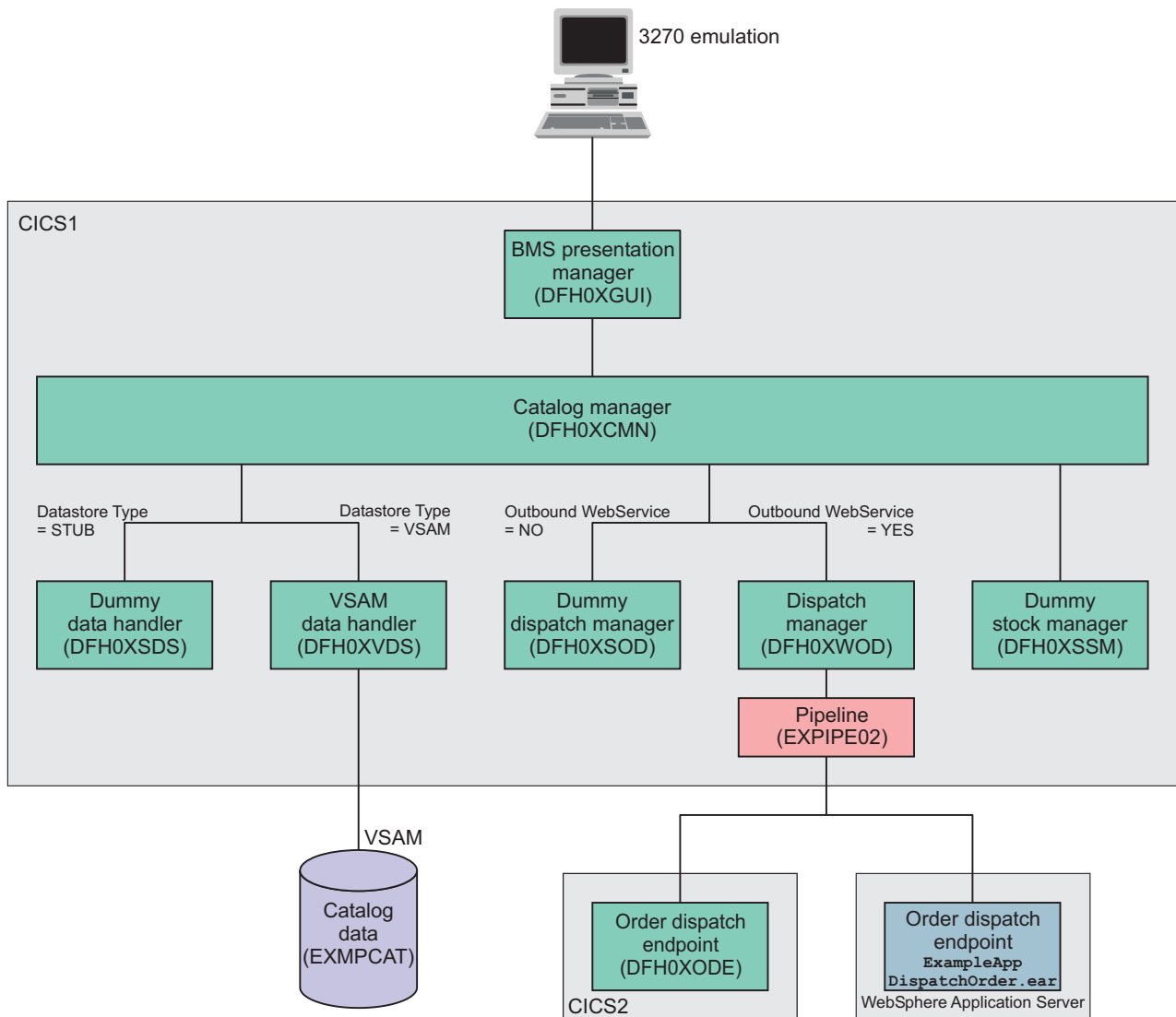


Figure 27. Structure of the base application

The components of the base application are:

1. A BMS presentation manager (DFH0XGUI) that supports a 3270 terminal or emulator, and that interacts with the main catalog manager program.
2. A catalog manager program (DFH0XCMN) that is the core of the example application, and that interacts with several back-end components.
3. The back-end components are:
 - A data handler program that provides the interface between the catalog manager program and the data store. The base application provides two versions of this program. They are the VSAM data handler program (DFH0XVDS), which stores data in a VSAM data set; and a dummy data handler (DFH0XSDS), which does not store data, but simply returns valid responses to its caller. Configuration options let you choose between the two programs.
 - A dispatch manager program that provides an interface for dispatching an order to a customer. Again, configuration options let you choose between the two versions of this program: DFH0XWOD is a Web service requester that

invokes a remote order dispatch end point, and DFHX0SOD is a dummy program that simply returns valid responses to its caller.

There are two equivalent order dispatch endpoints: DFHX0CODE is a CICS service provider program; `ExampleAppDispatchOrder.ear` is an enterprise archive that can be deployed in WebSphere Application Server or similar environments.

- A dummy stock manager program (DFHX0SSM) that returns valid responses to its caller, but takes no other action.

BMS presentation manager

The presentation manager is responsible for all interactions with the end user via 3270 BMS panels. No business decisions are made in this program.

The BMS presentation manager can be used in two ways:

- As part of the base application.
- As a CICS Web service client that communicates with the base application using SOAP messages.

Data handler

The data handler provides the interface between the catalog manager and the data store.

The example application provides two versions of the data handler:

- The first version uses a VSAM file as the data store.
- The second version is a dummy program that always returns the same data on an inquire and does not store the results of any update requests.

Dispatch manager

The dispatch manager is responsible for dispatching the order to the customer once the order has been confirmed.

The example application provides two versions of the dispatch manager program:

- The first version is a dummy program that returns a correct response to the caller, but takes no other action.
- The second version is a Web service requester program that makes a request to the endpoint address defined in the configuration file.

Order dispatch endpoint

The order dispatch program is a Web service provider program that is responsible for dispatching the item to the customer.

In the example application, the order dispatcher is a dummy program that returns a correct response to the caller, but takes no other action. It makes it possible for all configurations of the example Web services to be operable.

Stock manager

The stock manager is responsible for managing the replenishment of the stock.

In the example program, the stock manager is a dummy program that returns a correct response to the caller, but takes no other action.

Application configuration

The example application includes a program that lets you configure the base application.

Installing and setting up the base application

Before you can run the base application you must define and populate two VSAM data sets, and install two transaction definitions.

Creating and defining the VSAM data sets

The example application uses two KSDS VSAM data sets to be defined and populated. One data set contains configuration information for the example application. The other contains the sales catalog.

1. Locate the JCL to create the VSAM data sets. During CICS installation, the JCL is placed in the *hlq.SDFHINST* library:
 - Member DFH\$ECNF contains the JCL to generate the configuration data set.
 - Member DFH\$ECAT contains the JCL to generate the catalog data set.
2. Modify the JCL and access method services commands.
 - a. Supply a valid JOB card.
 - b. Supply a suitable high level qualifier for the data set names in the access method services commands. As supplied, the JCL uses a high level qualifier of HLQ.

The following command defines the catalog file:

```
DEFINE CLUSTER (NAME(hlq.EXMPLAPP.catname)-  
  TRK(1 1) -  
  KEYS(4 0) -  
  RECORDSIZE(80,80) -  
  SHAREOPTIONS(2 3) -  
  INDEXED -  
  ) -  
  DATA (NAME(hlq.EXMPLAPP.catname.DATA) -  
  ) -  
  INDEX (NAME(hlq.EXMPLAPP.catname.INDEX) -  
  )
```

where

- *hlq* is a high level qualifier of your choice
- *catname* is a name of your choice. The name used in the example application as supplied is EXMPCAT.

.

The following command defines the configuration file:

```
DEFINE CLUSTER (NAME(hlq.EXMPLAPP.EXMPCONF)-  
  TRK(1 1) -  
  KEYS(9 0) -  
  RECORDSIZE(350,350) -  
  SHAREOPTIONS(2 3) -  
  INDEXED -  
  ) -  
  DATA (NAME(hlq.EXMPLAPP.EXMPCONF.DATA) -  
  ) -  
  INDEX (NAME(hlq.EXMPLAPP.EXMPCONF.INDEX) -  
  )
```

where *hlq* is a high level qualifier of your choice.

3. Run both jobs to create and populate the data sets.

4. Use the CEDA transaction to create a FILE definition for the catalog file.
 - a. Enter the following: CEDA DEF FILE(EXMPCAT)G(EXAMPLE). Alternatively, you can copy the FILE definition from CICS supplied group DFH\$EXBS.
 - b. Enter the following additional attributes:


```
DSNAME(hlg.EXMPLAPP.EXMPCAT)
ADD(YES)
BROWSE(YES)
DELETE(YES)
READ(YES)
UPDATE(YES)
```
 - c. Use the default values for all other attributes.
5. Use the CEDA transaction to create a FILE definition for the configuration file.
 - a. Enter the following: CEDA DEF FILE(EXMPCONF) G(EXAMPLE). Alternatively, you can copy the FILE definition from CICS supplied group DFH\$EXBS.
 - b. Enter the following additional attributes:


```
DSNAME(hlg.EXMPLAPP.EXMPCONF)
ADD(YES)
BROWSE(YES)
DELETE(YES)
READ(YES)
UPDATE(YES)
```
 - c. Use the default values for all other attributes.

Defining the 3270 interface

The example application is supplied with a 3270 user interface to run the application and to customize it. The user interface consists of two transactions, EGUI and ECFG. A third transaction, ECLI, is used for the CICS Web service client.

1. Use the CEDA transaction to create TRANSACTION definitions for the transactions.
 - a. To define transaction EGUI, enter the following: CEDA DEF TRANS(EGUI) G(EXAMPLE) PROG(DFH0XGUI).
 - b. To define transaction ECFG, enter the following: CEDA DEF TRANS(ECFG) G(EXAMPLE) PROG(DFH0XCFG)
 - c. Optional: To define transaction ECLI, enter the following: CEDA DEF TRANS(ECLI) G(EXAMPLE) PROG(DFH0XCUI)

Use the default values for all other attributes.

Note: The correct operation of the example application does not depend on the names of the transactions, so you can use different names if you wish. Alternatively, you can copy the TRANSACTION definitions for EGUI and ECFG from CICS supplied group DFH\$EXBS, and the definition for ECLI from group DFH\$EXWS.

2. Optional: If you do not wish to use program autoinstall, use the CEDA transaction to create PROGRAM definitions for the base application programs and MAPSET definitions for the BMS maps.
 - a. Define MAPSET resource definitions for the BMS maps in members DFH0XS1, DFH0XS2, and DFH0XS3. For details of what is in each member, see “Components of the base application” on page 227.

- b. Define PROGRAM resource definitions, using the command `CEDA DEF PROG(program) G(EXAMPLE)`. You should create definitions for the following COBOL programs:

Table 6. SDFHSAMP members containing COBOL source for the base application.

Member name	Description
DFH0XCFG	Program invoked by transaction ECFG to read and update the VSAM configuration file
DFH0XCMN	Controller program for the catalog application. All requests pass through it.
DFH0XGUI	Program invoked by transaction EGUI to manage the sending of the BMS maps to the terminal user and the receiving of the maps from the terminal user. It links to program DFH0XCMN.
DFH0XODE	One of two versions of the endpoint for the order dispatch Web service. This is the version that runs in CICS. It simply sets the text "order in dispatch" in the return COMMAREA.
DFH0XSDS	A <i>stubbed</i> or dummy version of the data store program that allows the application to work when the VSAM catalog file has not been set up. It uses data defined in the program rather than data stored in a VSAM file.
DFH0XSOD	A stubbed version of the order dispatch program. It sets the return code in the COMMAREA to 0 and returns to its caller. It is used when outbound Web services are not required.
DFH0XSSM	A stubbed version of the stock manager (replenishment) program. It sets the return code in the COMMAREA to 0 and returns to its caller.
DFH0XVDS	The VSAM version of the data store program. It accesses the VSAM file to perform reads and updates of the catalog.
DFH0XWOD	The Web service version of the order dispatch program. It issues an EXEC CICS INVOKE WEBSERVICE to make an outbound Web service call to an order dispatcher

Use the default values for all other attributes.

- c. Optional: To define COBOL program DFH0XCUI, enter the following: `CEDA DEF PROG(DFH0XCUI) G(EXAMPLE)`. Use the default values for all other attributes. This program is required if you want to use transaction ECLI that starts the Web service client.

Completing the installation

To complete the installation, install the RDO group that contains your resource definitions.

Enter the following command at a CICS terminal: `CEDA I G(EXAMPLE)`.

The application is now ready for use.

Configuring the example application

The base application includes a transaction (ECFG) that you can use to configure the example application.

The configuration transaction uses mixed case information. You must use a terminal that can handle mixed case information correctly.

The transaction lets you specify a number of aspects of the example application. These include:

- The overall configuration of the application, such as the use of Web services
- The network addresses used by the Web services components of the application
- The names of resources, such as the file used for the data store
- The names of programs used for each component of the application

The configuration transaction lets you replace CICS-supplied components of the example application with your own, without restarting the application.

1. Enter the transaction ECFG to start the configuration application. CICS displays the following screen:

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

      Datastore Type ==> VSAM           STUB|VSAM
Outbound WebService? ==> NO           YES|NO
      Catalog Manager ==> DFH0XCMN
      Data Store Stub ==> DFH0XSDS
      Data Store VSAM ==> DFH0XVDS
      Order Dispatch Stub ==> DFH0XSOD
Order Dispatch WebService ==> DFH0XWOD
      Stock Manager ==> DFH0XSSM
      VSAM File Name ==> EXMPCAT
Server Address and Port ==> myserver:99999
Outbound WebService URI ==> http://myserver:80/exampleApp/dispatchOrder
                        ==>
                        ==>
                        ==>
                        ==>
                        ==>

PF                               3  END                               12  CNCL
```

2. Complete the fields.

Datastore Type

Specify STUB if you want to use the Data Store Stub program.

Specify VSAM if you want to use the VSAM data store program.

Outbound WebService

Specify YES if you want to use a remote Web service for your Order Dispatch function, that is, if you want the catalog manager program to link to the Order Dispatch Web service program.

Specify NO if you want to use a stub program for your Order Dispatch function, that is, if you want the catalog manager program to link to the Order Dispatch Stub program.

Catalog Manager

Specify the name of the Catalog Manager program. The program supplied with the example application is DFH0XCMN.

Data Store Stub

If you specified STUB in the **Datastore Type** field, specify the name of the Data Store Stub program. The program supplied with the example application is DFH0XSDS.

Data Store VSAM

If you specified VSAM in the **Datastore Type** field, specify the name of the VSAM data store program. The program supplied with the example application is DFH0XVDS.

Order Dispatch Stub

If you specified NO in the **Outbound WebService** field, specify the name of the Order Dispatch Stub program. The program supplied with the example application is DFH0XSOD.

Order Dispatch WebService

If you specified YES in the **Outbound WebService** field, specify the name of the program that functions as a service requester. The program supplied with the example application is DFH0XWOD.

Stock Manager

Specify the name of the Stock Manager program. The program supplied with the example application is DFH0XSSM.

VSAM File Name

If you specified VSAM in the **Datastore Type** field, specify the name of the CICS FILE definition. The name used in the example application as supplied is EXMPCAT.

Server Address and Port

If you are using the CICS Web service client, specify the IP address and port of the system on which the example application is deployed as a Web service.

Outbound WebService URI

If you specified YES in the **Outbound WebService** field, specify the location of the Web service that implements the dispatch order function. If you are using the supplied CICS endpoint set this to: `http://myserver:myport/exampleApp/dispatchOrder` where *myserver* and *myport* are your CICS server address and port respectively.

Running the example application with the BMS interface

The base application can be invoked using its BMS interface.

1. Enter transaction EGUI from a CICS terminal. The example application displays the following menu:

```

CICS EXAMPLE CATALOG APPLICATION - Main Menu

Select an action, then press ENTER

Action . . . . 1. List Items
                2. Order Item Number ____
                3. Exit

F3=EXIT  F12=CANCEL

```

The options on the menu enable you to list the items in the catalog, order an item, or exit the application.

2. Type 1 and press ENTER to select the LIST ITEMS option. The application displays a list of items in the catalog.

```

CICS EXAMPLE CATALOG APPLICATION - Inquire Catalog

Select a single item to order with /, then press ENTER

Item  Description                      Cost  Order
-----
0010  Ball Pens Black 24pk                 2.90  /
0020  Ball Pens Blue 24pk                  2.90  -
0030  Ball Pens Red 24pk                   2.90  -
0040  Ball Pens Green 24pk                  2.90  -
0050  Pencil with eraser 12pk               1.78  -
0060  Highlighters Assorted 5pk             3.89  -
0070  Laser Paper 28-lb 108 Bright 500/ream  7.44  -
0080  Laser Paper 28-lb 108 Bright 2500/case 33.54  -
0090  Blue Laser Paper 201b 500/ream         5.35  -
0100  Green Laser Paper 201b 500/ream        5.35  -
0110  IBM Network Printer 24 - Toner cart    169.56  -
0120  Standard Diary: Week to view 8 1/4x5 3/4 25.99  -
0130  Wall Planner: Eraseable 36x24          18.85  -
0140  70 Sheet Hard Back wire bound notepad  5.89  -
0150  Sticky Notes 3x3 Assorted Colors 5pk   5.35  -

F3=EXIT  F7=BACK  F8=FORWARD  F12=CANCEL

```

3. Type / in the **ORDER** column, and press ENTER to order an item. The application displays details of the item to be ordered.

CICS EXAMPLE CATALOG APPLICATION - Details of your order

Enter order details, then press ENTER

Item	Description	Cost	Stock	On Order
0010	Ball Pens Black 24pk	2.90	0011	000

Order Quantity: 5
User Name: CHRISB
Charge Dept: CICSDEV1

F3=EXIT F12=CANCEL

4. If there is sufficient stock to fulfil the order, enter the following information.
 - a. Complete the ORDER QUANTITY field. Specify the number of items you want to order.
 - b. Complete the USERID field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
 - c. Complete the CHARGE DEPT field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
5. Press ENTER to submit the order and return to the main menu.
6. Select the EXIT option to end the application.

Web service support for the example application

The Web service support extends the example application, providing a Web client front end and two versions of a Web service endpoint for the order dispatcher component.

The Web client front end and one version of the Web service endpoint are supplied as enterprise archives (EARs) that will run in the following environments:

- WebSphere Application Server Version 5 Release 1 or later
- WebSphere Studio Application Developer Version 5 Release 1 or later with a WebSphere unit test environment
- WebSphere Studio Enterprise Developer Version 5 Release 1 or later with a WebSphere unit test environment

The second version of the Web service endpoint is supplied as a CICS service provider application program (DFH0XODE).

Figure 28 on page 209 shows one configuration of the example application.

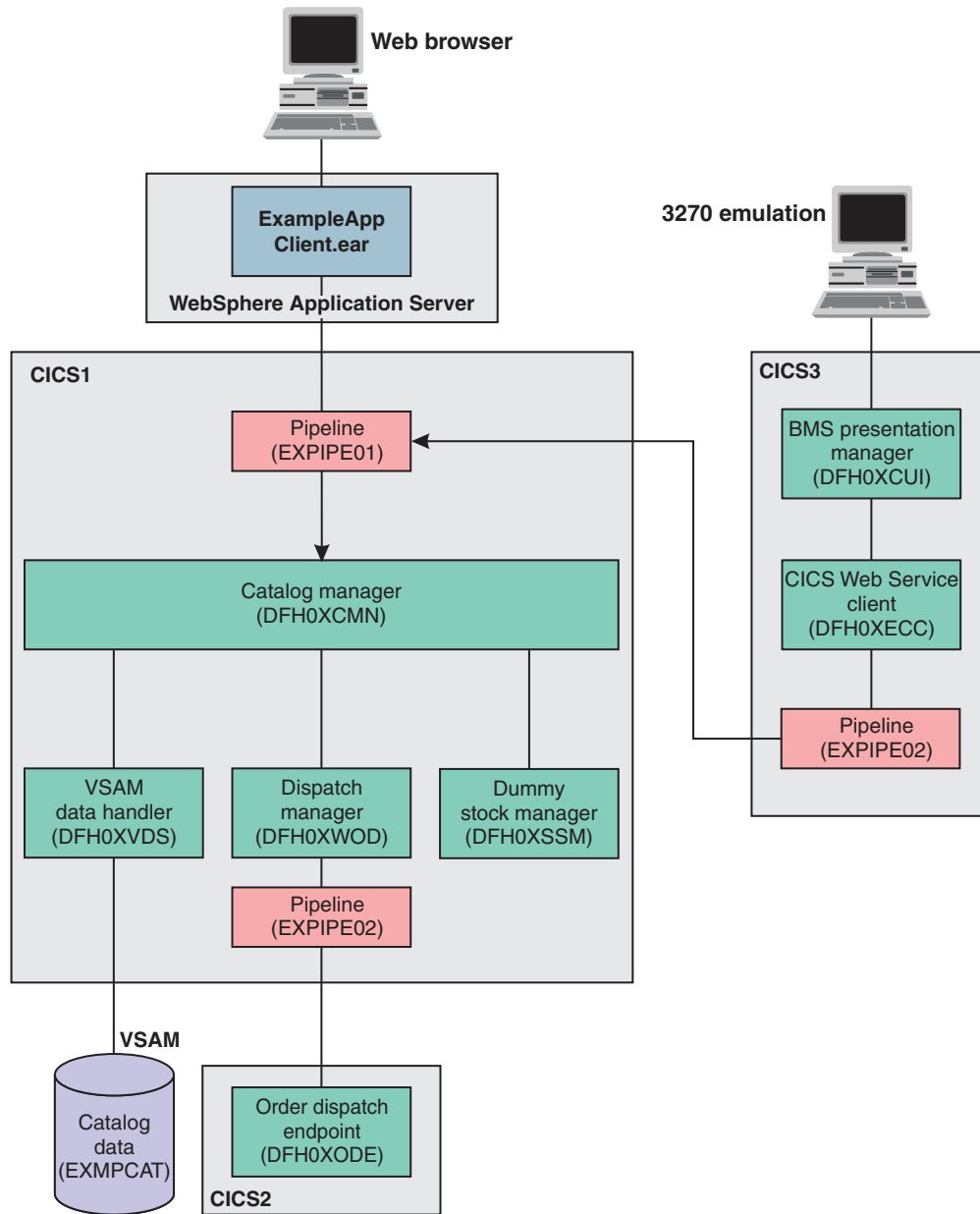


Figure 28. The example application configured as a Web service provider

In this configuration, the application is accessed through two different clients:

- A Web browser client connected to WebSphere Application Server, in which ExampleAppClient.ear is deployed.
- CICS Web service client DFH0XECC. This client uses the same BMS presentation logic as the base application but uses module DFH0XCUI instead of DFH0XGUI.

Figure 29 on page 210 shows another way to configure the example application as a Web service.

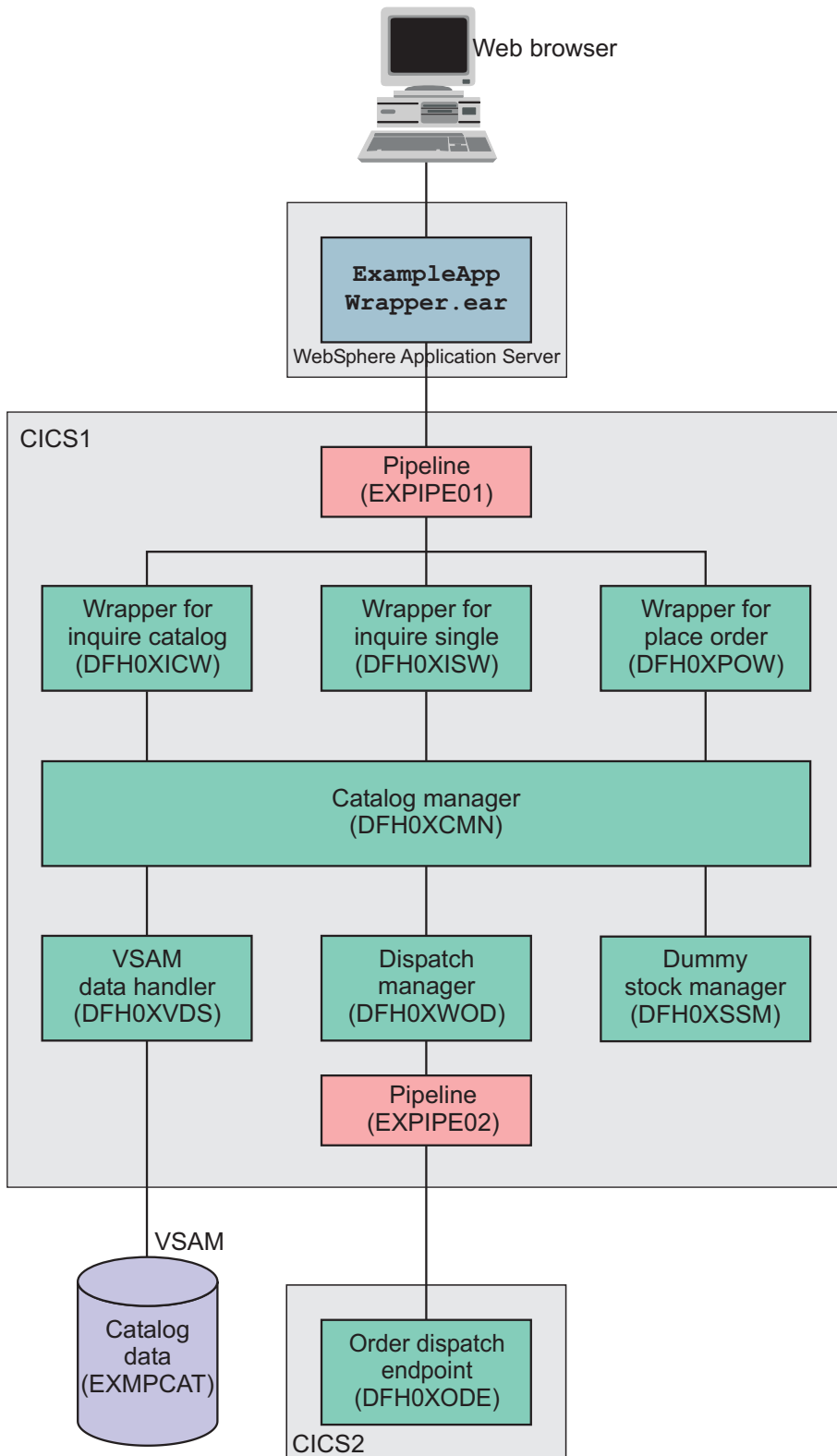


Figure 29. Alternate Web service provider configuration

In this configuration, the Web browser client is connected to WebSphere Application Server, in which ExampleAppWrapper.ear is deployed. In CICS, three wrapper

applications (for the inquire catalog, inquire single, and place order functions) are deployed as service provider applications. They in turn link to the base application.

Configuring code page support

As supplied, the example application uses two coded character sets. You must configure your system to enable data conversion between the two character sets.

The coded character sets used in the example application are:

CCSID Description

037 EBCDIC Group 1: USA, Canada (z/OS), Netherlands, Portugal, Brazil, Australia, New Zealand)

1208 UTF-8 Level 3

Add the following statements to the conversion image for your z/OS system:

```
CONVERSION 037,1208;  
CONVERSION 1208,037;
```

For more information, see *../dfhxa/dfha1/dfha1p9.dita#dfhp90dthe CICS Installation Guide*.

Defining the Web service client and wrapper programs

If you are not using program autoinstall, you need to define resource definitions for the Web service client and wrapper programs.

1. Define PROGRAM resource definitions for the wrapper programs, using the command `CEDA DEF PROG(program) G(EXAMPLE)`. You should create definitions for the following COBOL programs:

Table 7. SDFHSAMP members containing COBOL source code for the wrapper modules

Member name	Description
DFH0XICW	Wrapper program for the inquireCatalog service.
DFH0XISW	Wrapper program for the inquireSingle service.
DFH0XPOW	Wrapper program for the purchaseOrder service.

2. Define a PROGRAM resource definition for the Web services client program DFH0XECC, using the command `CEDA DEF PROG(DFH0XECC) G(EXAMPLE)`. This is a COBOL program. You can use default values for all of the other attributes.

Installing Web service support

Before you can run the Web service support for the example application, you must create two HFS directories, and create and install a number of CICS resource definitions.

Creating the HFS directories

Web service support for the example application requires a *shelf directory* and a *pickup directory* in the Hierarchical File System (HFS).

The shelf directory is used to store the Web service binding files that are associated with WEBSERVICE resources. Each WEBSERVICE resource is, in turn, associated with a PIPELINE. The shelf directory is managed by the PIPELINE resource and

you should not modify its contents directly. Several PIPELINES can use the same shelf directory, as CICS ensures a unique directory structure beneath the shelf directory for each PIPELINE.

The pickup directory is the directory that contains the Web service binding files associated with a PIPELINE. When a PIPELINE is installed, or in response to a PERFORM PIPELINE SCAN command, information in the binding files is used to dynamically create the WEBSERVICE and URIMAP definitions associated with the PIPELINE.

The example application uses `/var/cicsts` for the shelf directory.

A pipeline will read in an XML pipeline configuration file at install time. It is therefore also useful to define a directory in which to store these.

Creating the PIPELINE definition

The complete definition of a pipeline consists of a PIPELINE resource and a pipeline configuration file. The file contains the details of the message handlers that will act on Web service requests and responses as they pass through the pipeline.

The example application uses the CICS-supplied SOAP 1.1 handler to deal with the SOAP envelopes of inbound and outbound requests. CICS provides sample pipeline configuration files which you can use in your service provider and service requester.

More than one WEBSERVICE can share a single PIPELINE, therefore you need define only one pipeline for the inbound requests of the example application. You must, however, define a second PIPELINE for the outbound requests as a single PIPELINE cannot be configured to be both a provider and requester pipeline at the same time.

1. Use the CEDA transaction to create a PIPELINE definition for the service provider.
 - a. Enter the following: `CEDA DEF PIPE(EXPIPE01) G(EXAMPLE)`. Alternatively, you can copy the PIPELINE definition from CICS supplied group DFH\$EXWS.
 - b. Enter the following additional attributes:

```
STATUS(Enabled)
CONFIGFILE(/usr/lpp/cicsts
           /samples/pipelines/basicsoap11provider.xml)
SHELF(var/cicsts)
WSDIR(/usr/lpp/cicsts/samples/webservices/wsbinding/provider/)
```

Note that the HFS entries are case sensitive and assume a default CICS HFS install root of `/usr/lpp/cicsts`.

2. Use the CEDA transaction to create a PIPELINE definition for the service requester.
 - a. Enter the following: `CEDA DEF PIPE(EXPIPE02) G(EXAMPLE)`. Alternatively, you can copy the PIPELINE definition from CICS supplied group DFH\$EXWS.
 - b. Enter the following additional attributes:

```
STATUS(Enabled)
CONFIGFILE(/usr/lpp/cicsts
           /samples/pipelines/basicsoap11requester.xml)
SHELF(var/cicsts)
WSDIR(/usr/lpp/cicsts/samples/webservices/wsbinding/requester/)
```


Note that the HFS entries are case sensitive and assume a default CICS HFS install root of /usr/lpp/cicsts.

Creating a TCPIPSERVICE

As the client connects to your Web services over an HTTP transport you must define a TCPIPSERVICE to receive the inbound HTTP traffic.

Use the CEDA transaction to create a TCPIPSERVICE definition to handle inbound HTTP requests.

1. Enter the following: `CEDA DEF TCPIPSERVICE(EXMPPORT) G(EXAMPLE)`.
Alternatively, you can copy the TCPIPSERVICE definition from CICS supplied group DFH\$EXWS.
2. Enter the following additional attributes:
`URM(NONE)`
`PORTNUMBER(port)` where *port* is an unused port number in your CICS system.
`PROTOCOL(HTTP)`
`TRANSACTION(CWXN)`
3. Use the default values for all other attributes.

Dynamically installing the WEBSERVICE and URIMAP resources

Each function exposed as a Web service requires a WEBSERVICE resource to map between the incoming XML of the SOAP BODY and the COMMAREA interface of the program, and a URIMAP resource that routes incoming requests to the correct PIPELINE and WEBSERVICE. Although you can use RDO to define and install your WEBSERVICE and URIMAP resources, you can also have CICS create them dynamically when you install a PIPELINE resource.

Install the PIPELINE resources. Use the following commands:

```
CEDA INSTALL PIPELINE(EXPIPE01) G(EXAMPLE)
CEDA INSTALL PIPELINE(EXPIPE02) G(EXAMPLE)
```

When you install each PIPELINE resource, CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory). For each Web service binding file in the directory, that is for each file with the `.wsbind` suffix, CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is newer than the existing resources.

When the PIPELINE is later disabled and discarded all associated WEBSERVICE and URIMAP resources will also be discarded.

If you have already installed the PIPELINE, use the `PERFORM PIPELINE SCAN` command to initiate the scan of the PIPELINE's pickup directory.

When you have installed the PIPELINES, the following WEBSERVICES and their associated URIMAPs will be installed in your system:

```
dispatchOrder
dispatchOrderEndpoint
inquireCatalog
inquireSingle
placeOrder
```

The names of the WEBSERVICES are derived from the names of the Web service binding files; the names of the URIMAPs are generated dynamically. You can view the resources with a `CEMT INQUIRE WEBSERVICE` command:

```

I WEBS
STATUS: RESULTS - OVERTYPE TO MODIFY
Webs(dispatchOrder          ) Pip(EXPIPE02)
   Ins                               Dat(20041203)
Webs(dispatchOrderEndpoint  ) Pip(EXPIPE01)
   Ins Uri (£539140 ) Pro (DFH0XODE) Com   Dat(20041203)
Webs(inquireCatalog         ) Pip(EXPIPE01)
   Ins Uri (£539141 ) Pro (DFH0XCMN) Com   Dat(20041203)
Webs(inquireSingle          ) Pip(EXPIPE01)
   Ins Uri (£539142 ) Pro (DFH0XCMN) Com   Dat(20041203)
Webs(placeOrder             ) Pip(EXPIPE01)
   Ins Uri (£539150 ) Pro (DFH0XCMN) Com   Dat(20041203)

```

The display shows the names of the PIPELINE, the URIMAP, and the target program that is associated with each WEBSERVICE. Note that in this example, there is no URIMAP or target program displayed for WEBSERVICE(dispatchOrder) because the WEBSERVICE is for an outbound request. WEBSERVICE(dispatchOrderEndpoint) represents the local CICS implementation of the dispatch order service.

Creating the WEBSERVICE resources with RDO

As an alternative to using the PIPELINE scanning mechanism to install WEBSERVICE resources, you can create and install them using Resource Definition Online (RDO).

Important: If you use RDO to define the WEBSERVICE and URIMAP resources, you must ensure that their Web service binding files are **not** in the PIPELINE's pickup directory.

1. Use the CEDA transaction to create a WEBSERVICE definition for the inquire catalog function of the example application.
 - a. Enter the following: CEDA DEF WEBSERVICE(EXINQCWS) G(EXAMPLE).
 - b. Enter the following additional attributes:

```

PIPELINE(EXPIPE01)
WSBIND(/usr/lpp/cicsts/samples
       /webservices/wsbind/provider/inquireCatalog.wsbind)

```

2. Repeat the preceding step for each of the following functions of the example application.

Function	WEBSERVICE name	PIPELINE attribute	WSBIND attribute
INQUIRE SINGLE ITEM	EXINQSWS	EXPIPE01	/usr/lpp/cicsts/samples /webservices/wsbind /provider/inquireSingle.wsbind
PLACE ORDER	EXORDRWS	EXPIPE01	/usr/lpp/cicsts/samples /webservices/wsbind /provider/placeOrder.wsbind
DISPATCH STOCK	EXODRQWS	EXPIPE02	/usr/lpp/cicsts/samples /webservices/wsbind /requester/dispatchOrder.wsbind
DISPATCH STOCK endpoint (optional)	EXODEPWS	EXPIPE01	/usr/lpp/cicsts/samples /webservices/wsbind /provider/dispatchOrderEndpoint.wsbind

Creating the URIMAP resources with RDO

As an alternative to using the PIPELINE scanning mechanism to install URIMAP resources, you can create and install them using Resource Definition Online (RDO).

Important: If you use RDO to define the WEBSERVICE and URIMAP resources, you must ensure that their Web service binding files are **not** in the PIPELINE's pickup directory.

1. Use the CEDA transaction to create a URIMAP definition for the inquire catalog function of the example application.
 - a. Enter the following: CEDA DEF URIMAP(INQCURI) G(EXAMPLE).
 - b. Enter the following additional attributes:

```
USAGE(PIPELINE)
HOST(*)
PATH(/exampleApp/inquireCatalog)
TCPIPSERVICE(SOAPPORT)
PIPELINE(EXPIPE01)
WEBSERVICE(EXINQWS)
```

2. Repeat the preceding step for each of the remaining functions of the example application. Use the following names for your URIMAPs:

Function	URIMAP name
INQUIRE SINGLE ITEM	INQSURI
PLACE ORDER	ORDRURI
DISPATCH STOCK	Not required
DISPATCH STOCK endpoint (optional)	ODEPURI

- a. Specify the following distinct attributes for each URIMAP:

Function	URIMAP name	PATH	WEBSERVICE
INQUIRE SINGLE ITEM	INQSURI	/exampleApp/inquireSingle	EXINQSWs
PLACE ORDER	ORDRURI	/exampleApp/placeOrder	EXORDRWS
DISPATCH STOCK endpoint (optional)	ODEPURI	/exampleApp/dispatchOrder	EXODEPWS

- b. Enter the following additional attributes, which are the same for all the URIMAPs:

```
USAGE(PIPELINE)
HOST(*)
TCPIPSERVICE(SOAPPORT)
PIPELINE(EXPIPE01)
```

Completing the installation

To complete the installation, install the RDO group that contains your resource definitions.

Enter the following command at a CICS terminal: CEDA I G(EXAMPLE).

The application is now ready for use.

Configuring the Web client

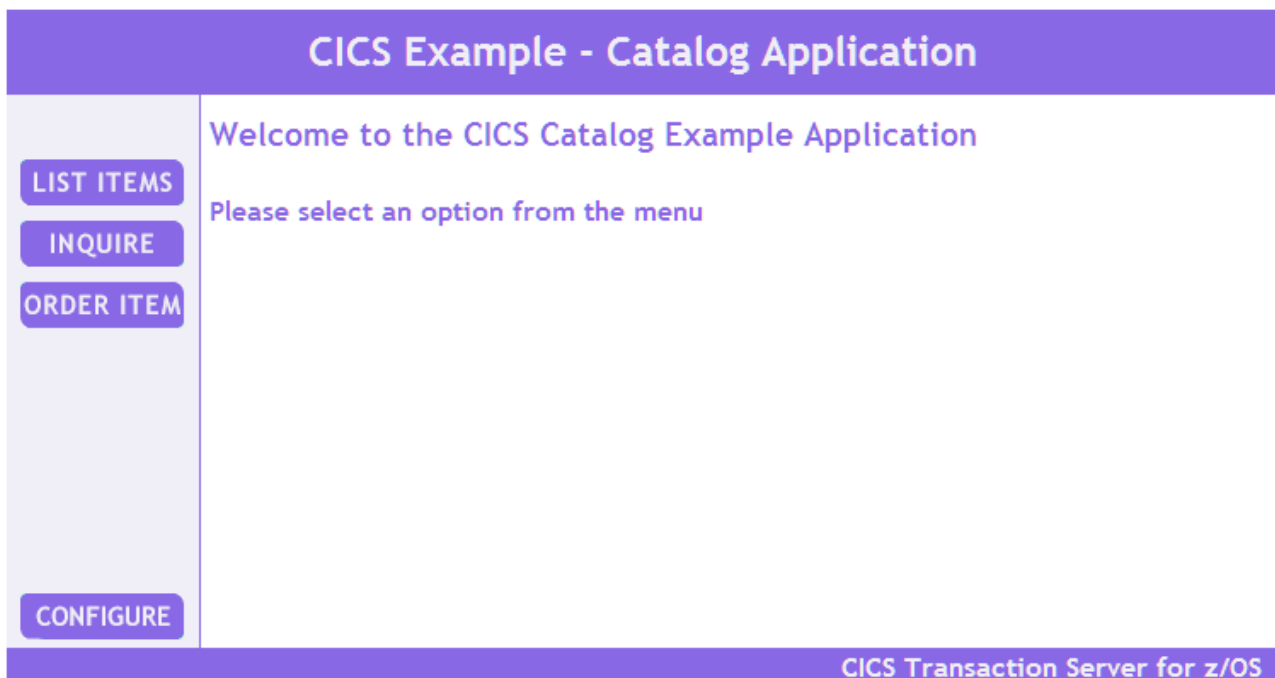
Before you can use the Web client, you must deploy the appropriate enterprise archive (EAR) for the client into one of the supported environments and configure it to call the appropriate end points in your CICS system.

The supported environments for the ExampleAppClient.ear client application are:

- WebSphere Application Server Version 5 Release 1
- WebSphere Studio Application Developer Version 5 Release 1 with a WebSphere unit test environment
- WebSphere Studio Enterprise Developer Version 5 Release 1 with a WebSphere unit test environment.

The supported environments for the ExampleAppClientV6.ear client application are:

- WebSphere Application Server Version 6
 - Rational® Application Developer Version 6 or later with a WebSphere unit test environment
 - WebSphere Developer for zSeries® Version 6 or later with a WebSphere unit test environment
1. Enter the following in your Web browser: `http://myserver:9080/ExampleAppClientWeb/`, where *myserver* is the hostname of the server on which the Web service client is installed. The example application displays the following page:



This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

2. Click the **CONFIGURE** button to bring up the configuration page. The configuration page is displayed.

CICS Example - Catalog Application

Configure Application

LIST ITEMS
INQUIRE
ORDER ITEM

Inquire Catalog Service Endpoint

Current	http://myCicsServer:9999/exampleApp/inquireCatalog
New	http://myCicsServer:9999/exampleApp/inquireCatalog

Inquire Item Service Endpoint

Current	http://myCicsServer:9999/exampleApp/inquireSingle
New	http://myCicsServer:9999/exampleApp/inquireSingle

Place Order Service Endpoint

Current	http://myCicsServer:9999/exampleApp/placeOrder
New	http://myCicsServer:9999/exampleApp/placeOrder

SUBMIT
RESET

BACK

CICS Transaction Server for z/OS

This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

3. Enter the new endpoints for the Web service. There are three endpoints to configure:
 - Inquire catalog
 - Inquire item
 - Place order
 - a. In the URLs replace the string 'myCicsServer' with the name of the system on which your CICS is running.
 - b. Replace the port number '9999' with the port number configured in the TCPIP SERVICE, in the example this to 30000.
4. Click the **SUBMIT** button.

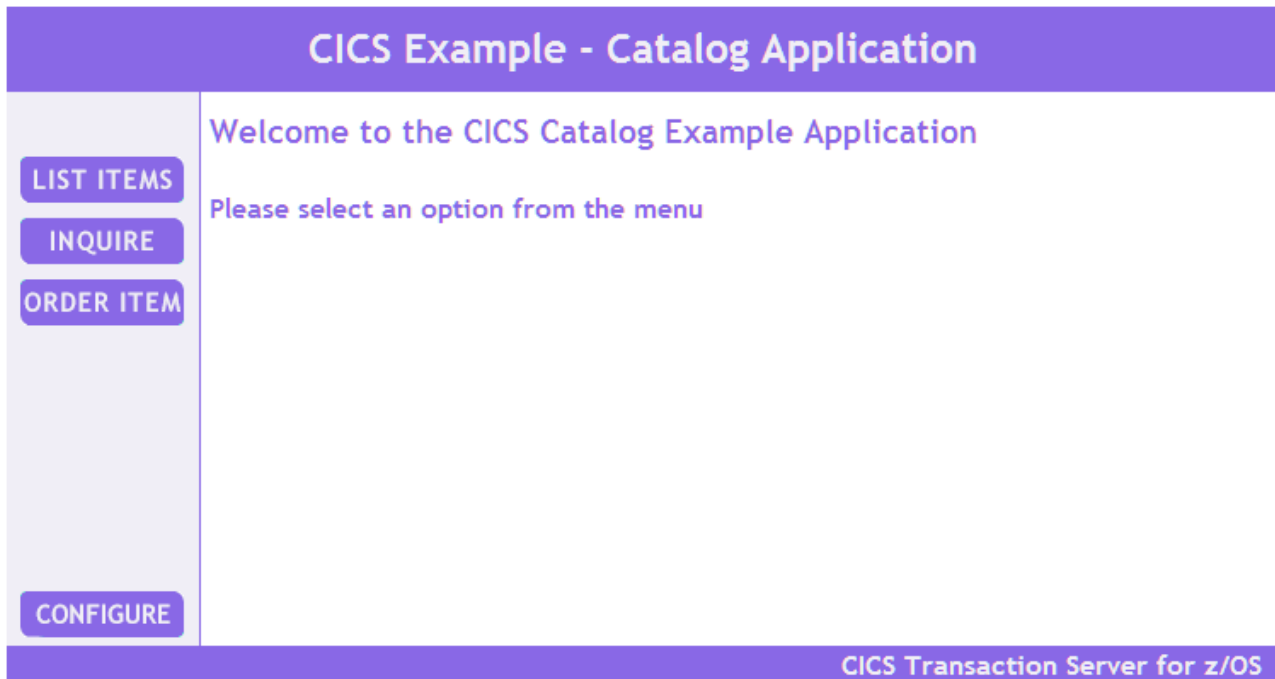
The Web application is now ready to run.

Note: The URL the Web services invoke is stored in an HTTP session. It is therefore necessary to repeat this configuration step each time a browser is first connected to the client.

Running the Web service enabled application

You can invoke the example application from a Web browser.

1. Enter the following in your Web browser: `http://myserver:9080/ExampleAppClientWeb/`, where *myserver* is the host name of the server on which the Web service client is installed. The example application displays the following page:



This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

2. Click the **INQUIRE** button. The example application displays the following page:

CICS Example - Catalog Application

Enter Catalog Item Reference Number

INQUIRE

ORDER ITEM

Start List From Item Number 0010

SUBMIT

BACK

CONFIGURE

CICS Transaction Server for z/OS

This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

3. Enter an item number, and click the **SUBMIT** button.

Tip: The base application is primed with item numbers in the sequence 0010, 0020, ... through 0210.

The application displays the following page, which contains a list of items in the catalog, starting with the item number that you entered.

CICS Example - Catalog Application

Item Details - Select Item to Place Order

LIST ITEMS

INQUIRE

ORDER ITEM

Item	Description	In Stock	On Order	Cost	Select
0010	Ball Pens Black 24pk	13	0	£2.90	<input type="radio"/>
0020	Ball Pens Blue 24pk	2	50	£2.90	<input type="radio"/>
0030	Ball Pens Red 24pk	38	0	£2.90	<input type="radio"/>
0040	Ball Pens Green 24pk	71	0	£2.90	<input checked="" type="radio"/>
0050	Pencil with eraser 12pk	70	0	£1.78	<input type="radio"/>
0060	Highlighters Assorted 5pk	11	40	£3.89	<input type="radio"/>
0070	Laser Paper 28-lb 108 Bright 500/ream	90	20	£7.44	<input type="radio"/>
0080	Laser Paper 28-lb 108 Bright 2500/case	25	0	£33.54	<input type="radio"/>
0090	Blue Laser Paper 20lb 500/ream	22	0	£5.35	<input type="radio"/>
0100	Green Laser Paper 20lb 500/ream	3	20	£5.35	<input type="radio"/>
0110	IBM Network Printer 24 - Toner cart	8	0	£169.56	<input type="radio"/>
0120	Standard Diary: Week to view 8 1/4x5 3/4	7	0	£25.99	<input type="radio"/>
0130	Wall Planner: Eraseable 36x24	3	0	£18.85	<input type="radio"/>
0140	70 Sheet Hard Back wire bound notepad	84	0	£5.89	<input type="radio"/>
0150	Sticky Notes 3x3 Assorted Colors 5pk	22	45	£5.35	<input type="radio"/>

SUBMIT

BACK

CONFIGURE

CICS Transaction Server for z/OS

This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

4. Select the item that you want to order.
 - a. Click the radio button in the **Select** column for the item you want to order.
 - b. Click the **SUBMIT** button.

The application displays the following page:

CICS Example - Catalog Application

Enter Order Details

LIST ITEMS

INQUIRE

Item Reference Number 0040

Quantity 001

User Name AUSER

Department Name CICS1

SUBMIT

BACK

CONFIGURE

CICS Transaction Server for z/OS

This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

5. To place an order, enter the following information.
 - a. Complete the Quantity field. Specify the number of items you want to order.
 - b. Complete the User Name field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
 - c. Complete the Department Name field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
 - d. Click the **SUBMIT** button.

The application displays the following page to confirm that the order has been placed:



This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

Deploying the example application

You can use the Web services assistant to deploy parts of the example application as a Web service. Although the application as supplied will work without performing this task, you will need to perform a similar task if you want to deploy your own applications to extend the example application.

Extracting the program interface

In order to deploy a program with the CICS Web services assistant, you must create a copybook that matches the program's COMMAREA or container interface.

In this example, the INQUIRE SINGLE ITEM function of the central Catalog Manager program (DFH0XCMN) will be deployed as a Web service. The interface to this program is a COMMAREA; the structure of the COMMAREA is defined in the copy book DFH0XCP1:

```
* Catalogue COMMAREA structure
   03 CA-REQUEST-ID          PIC X(6).
   03 CA-RETURN-CODE        PIC 9(2).
   03 CA-RESPONSE-MESSAGE  PIC X(79).
   03 CA-REQUEST-SPECIFIC  PIC X(911).
* Fields used in Inquire Catalog
   03 CA-INQUIRE-REQUEST  REDEFINES CA-REQUEST-SPECIFIC.
```

```

05 CA-LIST-START-REF      PIC 9(4).
05 CA-LAST-ITEM-REF      PIC 9(4).
05 CA-ITEM-COUNT         PIC 9(3).
05 CA-INQUIRY-RESPONSE-DATA PIC X(900).
05 CA-CAT-ITEM REDEFINES CA-INQUIRY-RESPONSE-DATA
    OCCURS 15 TIMES.
    07 CA-ITEM-REF        PIC 9(4).
    07 CA-DESCRIPTION     PIC X(40).
    07 CA-DEPARTMENT      PIC 9(3).
    07 CA-COST            PIC X(6).
    07 IN-STOCK          PIC 9(4).
    07 ON-ORDER          PIC 9(3).
*   Fields used in Inquire Single
03 CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC.
05 CA-ITEM-REF-REQ       PIC 9(4).
05 FILLER                PIC 9(4).
05 FILLER                PIC 9(3).
05 CA-SINGLE-ITEM.
    07 CA-SNGL-ITEM-REF   PIC 9(4).
    07 CA-SNGL-DESCRIPTION PIC X(40).
    07 CA-SNGL-DEPARTMENT PIC 9(3).
    07 CA-SNGL-COST      PIC X(6).
    07 IN-SNGL-STOCK     PIC 9(4).
    07 ON-SNGL-ORDER     PIC 9(3).
05 FILLER                PIC X(840).
*   Fields used in Place Order
03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
05 CA-USERID             PIC X(8).
05 CA-CHARGE-DEPT       PIC X(8).
05 CA-ITEM-REF-NUMBER   PIC 9(4).
05 CA-QUANTITY-REQ     PIC 9(3).
05 FILLER               PIC X(888).

```

The copybook defines 3 separate interfaces for the INQUIRE CATALOG, INQUIRE SINGLE ITEM and the PLACE ORDER functions, which are overlaid on one another in the copybook. However, the DFHLS2WS utility does not support the REDEFINES statement. Therefore you must extract from the combined copybook just those sections that relate to the inquire single function:

```

*   Catalogue COMMAREA structure
03 CA-REQUEST-ID         PIC X(6).
03 CA-RETURN-CODE        PIC 9(2) DISPLAY.
03 CA-RESPONSE-MESSAGE   PIC X(79).
*   Fields used in Inquire Single
03 CA-INQUIRE-SINGLE.
05 CA-ITEM-REF-REQ       PIC 9(4) DISPLAY.
05 FILLER                PIC X(4) DISPLAY.
05 FILLER                PIC X(3) DISPLAY.
05 CA-SINGLE-ITEM.
    07 CA-SNGL-ITEM-REF   PIC 9(4) DISPLAY.
    07 CA-SNGL-DESCRIPTION PIC X(40).
    07 CA-SNGL-DEPARTMENT PIC 9(3) DISPLAY.
    07 CA-SNGL-COST      PIC X(6).
    07 IN-SNGL-STOCK     PIC 9(4) DISPLAY.
    07 ON-SNGL-ORDER     PIC 9(3) DISPLAY.
05 FILLER                PIC X(840).

```

The redefined element CA-REQUEST-SPECIFIC has been removed and replaced by the section of the copybook that redefined it for the inquire single function. This copybook is now suitable for use with the Web service assistant.

This copybook is supplied with the example application as copybook DFH0XCP4.

Running the Web services assistant program DFHLS2WS

The CICS Web services assistant consists of two batch programs which can help you to transform existing CICS applications into Web services, and to enable CICS applications to use Web services provided by external providers. Program DFHLS2WS transforms a language structure to generate a Web service binding file and a Web service description.

1. Copy the supplied sample JCL to a suitable working file. The JCL is supplied in `samples/webservices/JCL/LS2WS`.
2. Add a valid JOB card to the JCL.
3. Code the parameters for DFHLS2WS. The required parameters for the INQUIRE SINGLE ITEM function of the example application are:

```
//INPUT.SYSUT1 DD *
LOGFILE=/u/exampleapp/wsbind/inquireSingle.log
PDSLIB=CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
PGMNAME=DFH0XCMN
PGMINT=COMMAREA
URI=exampleApp/inquireSingle
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsd1/inquireSingle.wsd1
*/
```

The parameters are as follows:

LOGFILE=/u/exampleapp/wsbind/inquireSingle.log

The file that is used to record diagnostic information from DFHLS2WS. The file is normally used only by IBM's software support organization.

PDSLIB=CICSHLQ.SDFHSAMP

The name of the partitioned data set (PDS) where the Web service assistant will look for copybooks that define the request and response structures. In the example this is SDFHSAMP of the CICS installed datasets.

REQMEM=DFH0XCP4

RESPMEM=DFH0XCP4

These parameters define the language structure for the request and the response to the program. In the example the request and the response have the same structure and are defined by the same copybook.

LANG=COBOL

The target program and the data structures are written in COBOL

PGMNAME=DFH0XCMN

The name of the target program that will be invoked when a Web service request is received.

PGMINT=COMMAREA

The target program is invoked with a COMMAREA interface.

URI=exampleApp/inquireSingle

The unique part of the URI that will be used in the generated Web service definition, and used to create the URIMAP resource that will map incoming requests to the correct Web service. The value specified will result in the service being available to external clients at:

`http://mycicsserver:myport/exampleApp/inquireSingle`

where *mycicsserver* and *myport* are the CICS server address and the port onto which this WEBSERVICE has been installed.

Note: The parameter does **not** have a leading '/'.

WSBIND=/u/exampleapp/wsbinding/inquireSingle.wsbinding

The location on HFS to which the Web service binding file will be written.

Note: If the file is to be used with the PIPELINE scanning mechanism it **must** have the extension `.wsbinding`.

WSDL=/u/exampleapp/wsd1/inquireSingle.wsdl

The location on HFS to which the file containing the generated Web service description will be written. It is good practice to use matching names for the Web service binding file and its corresponding Web service description.

Tip: Conventionally, files containing Web service descriptions have the extension `.wsdl`.

The Web services description provides the information that a client needs to access the Web service. It contains an XML schema definition of the request and response, and location information for the service.

4. Run the job. A Web service description and Web service binding file will be created in the locations specified.
5. Customize the service location in the Web service description. As generated, the `<service>` element contains the following:

```
<service name="DFH0XCMNService">
<port binding="tns:DFH0XCMNHTTPSoapBinding" name="DFH0XCMNPort">
<soap:address location="http://my-server:my-port/exampleApp/inquireSingle"/>
</port>
</service>
```

Before the Web service description can be published to clients, you must make the following changes:

- a. Replace *my-server* with the CICS server location.
- b. Replace *my-port* with the port number.

An example of the generated WSDL document

```
<?xml version="1.0" ?>
<definitions targetNamespace="http://www.DFH0XCMN.DFH0XCP4.com" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:reqns="http://www.DFH0XCMN.DFH0XCP4.Request.com" xmlns:resns="http://www.DFH0XCMN.DFH0XCP4.Response.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.DFH0XCMN.DFH0XCP4.com">
  <types>
    <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://www.DFH0XCMN.DFH0XCP4.Request.com" xmlns:tns="http://www.DFH0XCMN.DFH0XCP4.Request.com"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType abstract="false" block="#all" final="#all" mixed="false" name="ProgramInterface">
        <xsd:annotation>
          <xsd:documentation source="http://www.ibm.com/software/htp/cics/annotations">
            This schema was generated by the CICS Web services assistant.
          </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element name="ca_request_id" nillable="false">
            <xsd:simpletype>
              <xsd:annotation>
                <xsd:appinfo source="http://www.ibm.com/software/htp/cics/annotations">
                  #Thu Nov 03 11:55:26 GMT 2005 com.ibm.cics.wsdl.properties.synchronized=false
                </xsd:appinfo>
              </xsd:annotation>
            </xsd:simpletype>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <service name="DFH0XCMNService">
    <port binding="tns:DFH0XCMNHTTPSoapBinding" name="DFH0XCMNPort">
      <soap:address location="http://my-server:my-port/exampleApp/inquireSingle"/>
    </port>
  </service>
</definitions>
```

```

        <xsd:restriction base="xsd:string">
            <xsd:maxlength value="6"/>
            <xsd:whitespace value="preserve"/>
        </xsd:restriction>
    </xsd:simpletype>
</xsd:element>

```

.... most of the schema for the request is removed

```

    </xsd:sequence>
</xsd:complexType>
<xsd:element name="DFH0XCMNOperation" nillable="false" type="tns:ProgramInterface"/>
</xsd:schema>
<xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://www.DFH0XCMN.DFH0XCP4.Response.com" xmlns:tns="http://www.DFH0XCMN.DFH0XCP4.Response.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

```

... schema content for the reply is removed

```

    </xsd:schema>
</types>
<message name="DFH0XCMNOperationResponse">
    <part element="resns:DFH0XCMNOperationResponse" name="ResponsePart"/>
</message>
<message name="DFH0XCMNOperationRequest">
    <part element="reqns:DFH0XCMNOperation" name="RequestPart"/>
</message>
<porttype name="DFH0XCMNPort">
    <operation name="DFH0XCMNOperation">
        <input message="tns:DFH0XCMNOperationRequest" name="DFH0XCMNOperationRequest"/>
        <output message="tns:DFH0XCMNOperationResponse" name="DFH0XCMNOperationResponse"/>
    </operation>
</porttype>
<binding name="DFH0XCMNHTTPSoapBinding" type="tns:DFH0XCMNPort">
    <!-- This soap:binding indicates the use of SOAP 1.1 -->
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <!-- This soap:binding indicates the use of SOAP 1.2 -->
    <!-- <soap:binding style="document" transport="http://www.w3.org/2003/05/soap-http"/> -->
    <operation name="DFH0XCMNOperation">
        <soap:operation soapAction="" style="document"/>
        <input name="DFH0XCMNOperationRequest">
            <soap:body parts="RequestPart" use="literal"/>
        </input>
        <output name="DFH0XCMNOperationResponse">
            <soap:body parts="ResponsePart" use="literal"/>
        </output>
    </operation>
</binding>
<service name="DFH0XCMNService">
    <port binding="tns:DFH0XCMNHTTPSoapBinding" name="DFH0XCMNPort">
        <!-- This soap:address indicates the location of the Web service over HTTP.
            Please replace "my-server" with the TCPIP host name of your CICS region.
            Please replace "my-port" with the port number of your CICS TCIPSERVICE. -->
        <soap:address location="http://my-server:my-port/exampleApp/inquireSingles.log"/>
        <!-- This soap:address indicates the location of the Web service over HTTPS. -->
        <!-- <soap:address location="https://my-server:my-port/exampleApp/inquireSingles.log"/> -->
        <!-- This soap:address indicates the location of the Web service over MQSeries.
            Please replace "my-queue" with the appropriate queue name. -->
        <!-- <soap:address location="jms:/queue?destination=my-queue&connectionFactory=()&targetService=/exampleApp/inquireSingles.log&initialContextFactory=com.ibm.mq.jms.NoJndi" /> -->
    </port>
</service>
</definitions>

```

Deploying the Web services binding file

The Web services binding file created by DFHLS2WS is deployed into your CICS region dynamically when you install a PIPELINE resource.

When a PIPELINE scan command is issued, either explicitly via a CEMT P PIPELINE() SCAN or automatically during a PIPELINE installation, CICS scans the pickup directory to search for Web service binding files - that is, for file names with the .wsbind extension. For each binding file found, CICS determines whether to install a WEBSERVICE resource.

A URIMAP resource is also created to map the URI, as provided in the JCL, to the installed WEBSERVICE and the PIPELINE onto which the WEBSERVICE is installed. When a scanned WEBSERVICE is discarded the URIMAP associated with it is also discarded.

1. Modify PIPELINE(EXPIPE01), which is the PIPELINE definition for your provider pipeline. Change the WSDIR parameter to /u/exampleapp/wsbind. This pickup directory contains the Web service binding file that you generated with DFHLS2WS.
2. Copy any other Web service binding files used by the application to the same directory. In this example, the files to copy are:

```
inquireCatalog  
placeOrder
```

They are provided in directory /usr/lpp/cicsts/samples/webservices/wsbind/provider.

3. Install the PIPELINE. CICS will create a WEBSERVICE resource and a URIMAP resource from your Web service binding file.

Components of the base application

Table 8. SDFHSAMP members containing BMS maps

Member name	Description
DFH0XS1	BMS macros for the mapset consisting of the map (EXMENU) for the Main Menu screen and the map (EXORDR) for the Details of your order screen.
DFH0XS2	BMS macros for the mapset consisting of the map (EXINQC) for the Inquire Catalog screen.
DFH0XS3	BMS macros for the mapset consisting of the map (EXCONF) for the Configure CICS example catalog application screen.
DFH0XM1	Cobol copy book generated by assembling DFH0XS1. DFH0XGUI and DFH0XCUI include this copy book
DFH0XM2U	Cobol copy book generated by assembling DFH0XS2 and editing the result to include an indexed array structure for ease of copy book programming. DFH0XGUI and DFH0XCUI include this copy book.
DFH0XM3	Cobol copy book generated by assembling DFH0XS3. DFH0XCFG includes this copy book

Table 9. SDFHSAMP members containing COBOL source for the base application.

Member name	Description
DFH0XCFG	Program invoked by transaction ECFG to read and update the VSAM configuration file
DFH0XCMN	Controller program for the catalog application. All requests pass through it.
DFH0XGUI	Program invoked by transaction EGUI to manage the sending of the BMS maps to the terminal user and the receiving of the maps from the terminal user. It links to program DFH0XCMN.
DFH0XODE	One of two versions of the endpoint for the order dispatch Web service. This is the version that runs in CICS. It simply sets the text "order in dispatch" in the return COMMAREA.
DFH0XSDS	A <i>stubbed</i> or dummy version of the data store program that allows the application to work when the VSAM catalog file has not been set up. It uses data defined in the program rather than data stored in a VSAM file.
DFH0XSOD	A stubbed version of the order dispatch program. It sets the return code in the COMMAREA to 0 and returns to its caller. It is used when outbound Web services are not required.
DFH0XSSM	A stubbed version of the stock manager (replenishment) program. It sets the return code in the COMMAREA to 0 and returns to its caller.
DFH0XVDS	The VSAM version of the data store program. It accesses the VSAM file to perform reads and updates of the catalog.
DFH0XWOD	The Web service version of the order dispatch program. It issues an EXEC CICS INVOKE WEBSERVICE to make an outbound Web service call to an order dispatcher

Table 10. SDFHSAMP members containing COBOL copy books for the basic application

Member name	Description
DFH0XCP1	Defines a COMMAREA structure which includes the request and response for the inquire catalog, inquire single, and place order functions. Programs DFH0XCMN, DFH0XCUI, DFH0XECC, DFH0XGUI, DFH0XICW, DFH0XISW, DFH0XPOW, DFH0XSDS, and DFH0XVDS include this copy book.
DFH0XCP2	Defines a COMMAREA structure for the order dispatcher and stock manager modules. Programs DFH0XCMN, DFH0XSOD, DFH0XSSM, and DFH0XWOD include this copy book
DFH0XCP3	Defines a data structure for an inquire catalog request and response. Used as input to DFHLS2WS in order to produce inquireCatalog.wsd1 and inquireCatalog.wsbinding .
DFH0XCP4	Defines a data structure for an inquire single request and response. Used as input to DFHLS2WS in order to produce inquireSingle.wsd1 and inquireSingle.wsbinding.
DFH0XCP5	Defines a data structure for a place order request and response. Used as input to DFHLS2WS in order to produce placeOrder.wsd1 and placeOrder.wsbinding
DFH0XCP6	Defines a data structure for a dispatch order request and response. Used as input to DFHLS2WS in order to produce dispatchOrder.wsd1 and dispatchOrder.wsbinding
DFH0XCP7	Defines the data structure for a dispatch order request. Programs DFH0XODE and DFH0XWOD include this copy book

Table 10. SDFHSAMP members containing COBOL copy books for the basic application (continued)

Member name	Description
DFH0XCP8	Defines the data structure for a dispatch order response. Programs DFH0XODE and DFH0XWOD include this copy book.

Table 11. SDFHSAMP members containing COBOL source code for the Web service client application which runs in CICS

Member name	Description
DFH0XCUI	Program invoked by transaction ECLI to manage the sending of the BMS maps to the terminal user and the receiving of the maps from the terminal user. It links to program DFH0XECC.
DFH0XECC	Makes outbound Web service requests to the base application, using the EXEC CICS INVOKE WEBSERVICE command. The WEBSERVICE specified is one of the following: inquireCatalogClient inquireSingleClient placeOrderClient

Table 12. SDFHSAMP members containing COBOL copy books for the Web service client application which runs in CICS.. They are all generated by DFHWS2LS, and are included by program DFH0XECC.

Member name	Description
DFH0XCPA	Defines the data structure for the inquire catalog request.
DFH0XCPB	Defines the data structure for the inquire catalog response.
DFH0XCPC	Defines the data structure for the inquire single request.
DFH0XCPD	Defines the data structure for the inquire single response.
DFH0XCPE	Defines the data structure for the place order request.
DFH0XCPF	Defines the data structure for the place order response.

Table 13. SDFHSAMP members containing COBOL source code for the wrapper modules

Member name	Description
DFH0XICW	Wrapper program for the inquireCatalog service.
DFH0XISW	Wrapper program for the inquireSingle service.
DFH0XPOW	Wrapper program for the purchaseOrder service.

Table 14. SDFHSAMP members containing COBOL copy books for the wrapper modules

Member name	Description
DFH0XWC1	Defines the data structure for the inquire catalog request. Program DFH0XICW includes this copy book.
DFH0XWC2	Defines the data structure for the inquire catalog response. Program DFH0XICW includes this copy book.
DFH0XWC3	Defines the data structure for the inquire single request. Program DFH0XISW includes this copy book.
DFH0XWC4	Defines the data structure for the inquire single response. Program DFH0XISW includes this copy book.

Table 14. SDFHSAMP members containing COBOL copy books for the wrapper modules (continued)

Member name	Description
DFH0XWC5	Defines the data structure for the place order request. Program DFH0XPOW includes this copy book.
DFH0XWC6	Defines the data structure for the place order response. Program DFH0XPOW includes this copy book

Table 15. CICS Resource Definitions

Resource name	Resource type	Comment
EXAMPLE	CICS Resource definition group	CICS resource definitions required for the example application
EGUI	TRANSACTION	Transaction to invoke program DFH0XGUI to start the BMS interface to the application (Customizable)
ECFG	TRANSACTION	Transaction to invoke the program DFH0XCFG to start the example configuration BMS interface (Customizable)
EXMPCAT	FILE	File definition of the EXMPCAT VSAM file for the application catalog (Customizable)
EXMPCONF	FILE	File definition of the EXMPCONF application configuration file.

The catalog manager program

The catalog manager is the controlling program for the business logic of the example application, and all interactions with the example application pass through it.

To ensure that the program logic is simple, the catalog manager performs only limited type checking and error recovery.

The catalog manager supports a number of operations. Input and output parameters for each operation are defined in a single data structure, which is passed to and from the program in a COMMAREA.

COMMAREA structures

```
* Catalogue COMMAREA structure
  03 CA-REQUEST-ID          PIC X(6).
  03 CA-RETURN-CODE         PIC 9(2).
  03 CA-RESPONSE-MESSAGE   PIC X(79).
  03 CA-REQUEST-SPECIFIC   PIC X(911).
* Fields used in Inquire Catalog
  03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
    05 CA-LIST-START-REF    PIC 9(4).
    05 CA-LAST-ITEM-REF    PIC 9(4).
    05 CA-ITEM-COUNT       PIC 9(3).
    05 CA-INQUIRY-RESPONSE-DATA PIC X(900).
```

```

05 CA-CAT-ITEM REDEFINES CA-INQUIRY-RESPONSE-DATA
                OCCURS 15 TIMES.
07 CA-ITEM-REF PIC 9(4).
07 CA-DESCRIPTION PIC X(40).
07 CA-DEPARTMENT PIC 9(3).
07 CA-COST PIC X(6).
07 IN-STOCK PIC 9(4).
07 ON-ORDER PIC 9(3).
* Fields used in Inquire Single
03 CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC.
05 CA-ITEM-REF-REQ PIC 9(4).
05 FILLER PIC 9(4).
05 FILLER PIC 9(3).
05 CA-SINGLE-ITEM.
07 CA-SNGL-ITEM-REF PIC 9(4).
07 CA-SNGL-DESCRIPTION PIC X(40).
07 CA-SNGL-DEPARTMENT PIC 9(3).
07 CA-SNGL-COST PIC X(6).
07 IN-SNGL-STOCK PIC 9(4).
07 ON-SNGL-ORDER PIC 9(3).
05 FILLER PIC X(840).
* Fields used in Place Order
03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
05 CA-USERID PIC X(8).
05 CA-CHARGE-DEPT PIC X(8).
05 CA-ITEM-REF-NUMBER PIC 9(4).
05 CA-QUANTITY-REQ PIC 9(3).
05 FILLER PIC X(888).

* Dispatcher/Stock Manager COMMAREA structure
03 CA-ORD-REQUEST-ID PIC X(6).
03 CA-ORD-RETURN-CODE PIC 9(2).
03 CA-ORD-RESPONSE-MESSAGE PIC X(79).
03 CA-ORD-REQUEST-SPECIFIC PIC X(23).
* Fields used in Dispatcher
03 CA-DISPATCH-ORDER REDEFINES CA-ORD-REQUEST-SPECIFIC.
05 CA-ORD-ITEM-REF-NUMBER PIC 9(4).
05 CA-ORD-QUANTITY-REQ PIC 9(3).
05 CA-ORD-USERID PIC X(8).
05 CA-ORD-CHARGE-DEPT PIC X(8).
* Fields used in Stock Manager
03 CA-STOCK-MANAGER-UPDATE REDEFINES CA-ORD-REQUEST-SPECIFIC.
05 CA-STK-ITEM-REF-NUMBER PIC 9(4).
05 CA-STK-QUANTITY-REQ PIC 9(3).
05 FILLER PIC X(16).

```

Return codes

Each operation of the catalog manager can return a number of return codes.

Type	Code	Explanation
General	00	Function completed without error
Catalog file	20	Item reference not found
	21	Error opening, reading, or ending browse of catalog file
	22	Error updating file

Type	Code	Explanation
Configuration file	50	Error opening configuration file
	51	Data store type was neither STUB nor VSAM
	52	Outbound Web service switch was neither Y nor N
Remote Web service	30	The EXEC CICS INVOKE WEBSERVICE command returned an INVREQ condition
	31	The EXEC CICS INVOKE WEBSERVICE command returned an NOTFND condition
	32	The EXEC CICS INVOKE WEBSERVICE command returned a condition other than INVREQ or NOTFND
Application	97	Insufficient stock to complete order
	98	Order quantity was not a positive number
	99	DFH0XCMN received a COMMAREA in which the CA-REQUEST-ID field was not set to one of the following: 01INQC, 01INQS, or 01ORDR

INQUIRE CATALOG operation

This operation returns a list of up to 15 catalog items, starting with the item specified by the caller.

Input parameters

CA-REQUEST-ID

A string that identifies the operation. For the INQUIRE CATALOG command, the string contains "01INQC"

CA-LIST-START-REF

The reference number of the first item to be returned.

Output parameters

CA-RETURN-CODE

CA-RESPONSE-MESSAGE

A human readable string, containing "*num* ITEMS RETURNED" where *num* is the number of items returned.

CA-LAST-ITEM-REF

The reference number of the last item returned.

CA-ITEM-COUNT

The number of items returned.

CA-CAT-ITEM

An array containing the list of catalog items returned. The array has 15 elements; if fewer than 15 items are returned, the remaining array elements contain blanks.

INQUIRE SINGLE ITEM operation

This operation returns a single catalog item specified by the caller.

Input parameters**CA-REQUEST-ID**

A string that identifies the operation. For the INQUIRE SINGLE ITEM command, the string contains "01INQS"

CA-ITEM-REF-REQ

The reference number of the item to be returned.

Output parameters**CA-RETURN-CODE****CA-RESPONSE-MESSAGE**

A human readable string, containing RETURNED ITEM: REF=*item-reference*' where *item-reference* is the reference number of the returned item.

CA-SINGLE-ITEM

An array containing in its first element the returned catalog item.

PLACE ORDER operation

This operation places an order for a single item. If the required quantity is not available a message is returned to the user. If the order is successful, a call is made to the Stock Manager informing it what item has been ordered and the quantity ordered.

Input parameters**CA-REQUEST-ID**

A string that identifies the operation. For the PLACE ORDER operation, the string contains '01ORDR'

CA-USERID

An 8-character user ID which the application uses for dispatch and billing.

CA-CHARGE-DEPT

An 8-character department ID which the application uses for dispatch and billing.

CA-ITEM-REF-NUMBER

The reference number of the item to be ordered.

CA-QUANTITY-REQ

The number of items required.

Output parameters**CA-RETURN-CODE****CA-RESPONSE-MESSAGE**

A human readable string, containing 'ORDER SUCCESSFULLY PLACED'.

DISPATCH STOCK operation

This operation places a call to the stock dispatcher program, which in turn dispatches the order to the customer.

Input parameters

CA-ORD-REQUEST-ID

A string that identifies the operation. For the DISPATCH ORDER operation, the string contains '01DSP0'

CA-ORD-USERID

An 8-character user ID which the application uses for dispatch and billing.

CA-ORD-CHARGE-DEPT

An 8-character department ID which the application uses for dispatch and billing.

CA-ORD-ITEM-REF-NUMBER

The reference number of the item to be ordered.

CA-ORD-QUANTITY-REQ

The number of items required.

Output parameters

CA-ORD-RETURN-CODE

NOTIFY STOCK MANAGER operation

This operation takes details of the order that has been placed to decide if stock replenishment is necessary.

Input parameters

CA-ORD-REQUEST-ID

A string that identifies the operation. For the NOTIFY STOCK MANAGER operation, the string contains '01STK0'

CA-STK-ITEM-REF-NUMBER

The reference number of the item to be ordered.

CA-STK-QUANTITY-REQ

The number of items required.

Output parameters

CA-ORD-RETURN-CODE

File Structures and Definitions

The example application uses two VSAM files: the catalog file which contains the details of all items stocked and their stock levels, and the configuration file which holds user-selected options for the application.

Catalog file

The catalog file is a KSDS VSAM file which contains all information relating to the product inventory.

Records in the file have the following structure:

Name	COBOL data type	Description
WS-ITEM-REF-NUM	PIC 9(4)	Item reference number
WS-DESCRIPTION	PIC X(40)	Item description
WS-DEPARTMENT	PIC 9(3)	Department identification number
WS-COST	PIC ZZZ.99	Item price
WS-IN-STOCK	PIC 9(4)	Number of items in stock
WS-ON-ORDER	PIC 9(3)	Number of items on order

Configuration file

The configuration file is a KSDS VSAM file which contains information used to configure the example application.

The configuration file is a KSDS VSAM file with four distinct records.

Table 16. General information record

Name	COBOL data type	Description
PROGS-KEY	PIC X(9)	Key field for the general information record, containing 'EXMP-CONF'
filler	PIC X	
DATASTORE	PIC X(4)	A character string that specifies the type of data store program to be used. Values are: 'STUB' 'VSAM'
filler	PIC X	
DO-OUTBOUND-WS	PIC X	A character that specifies whether the dispatch manager is make an outbound Web service request. Values are: 'Y' 'N'
filler	PIC X	
CATMAN-PROG	PIC X(8)	The name of the catalog manager program
filler	PIC X	
DSSTUB-PROG	PIC X(8)	The name of the dummy data handler program
filler	PIC X	
DSVSAM-PROG	PIC X(8)	The name of the VSAM data handler program
filler	PIC X	
ODSTUB-PROG	PIC X(8)	The name of the dummy order dispatcher module
filler	PIC X	

Table 16. General information record (continued)

Name	COBOL data type	Description
ODWEBS-PROG	PIC X(8)	The name of the outbound Web service order dispatcher program
filler	PIC X	
STKMAN-PROG	PIC X(8)	The name of the stock manager program
filler	PIC X(10)	

Table 17. Outbound URL record

Name	COBOL data type	Description
URL-KEY	PIC X(9)	Key field for the general information record, containing 'OUTBNDURL'
filler	PIC X	
OUTBOUND-URL	PIC X(255)	Outbound URL for the order dispatcher Web service request

Table 18. Catalog file information record

Name	COBOL data type	Description
URL-FILE-KEY	PIC X(9)	Key field for the general information record, containing 'VSAM-NAME'
filler	PIC X	
CATALOG-FILE-NAME	PIC X(8)	Name of the CICS FILE resource used for the catalog file

Table 19. Server information record

Name	COBOL data type	Description
WS-SERVER-KEY	PIC X(9)	Key field for the server information record, containing 'WS-SERVER'
filler	PIC X	
CATALOG-FILE-NAME	PIC X(8)	For the CICS Web service client only, the IP address and port of the system on which the example application is deployed as a Web service

Bibliography

The CICS Transaction Server for z/OS library

The published information for CICS Transaction Server for z/OS is delivered in the following forms:

The CICS Transaction Server for z/OS Information Center

The CICS Transaction Server for z/OS Information Center is the primary source of user information for CICS Transaction Server. The Information Center contains:

- Information for CICS Transaction Server in HTML format.
- Licensed and unlicensed CICS Transaction Server books provided as Adobe Portable Document Format (PDF) files. You can use these files to print hardcopy of the books. For more information, see “PDF-only books.”
- Information for related products in HTML format and PDF files.

One copy of the CICS Information Center, on a CD-ROM, is provided automatically with the product. Further copies can be ordered, at no additional charge, by specifying the Information Center feature number, 7014.

Licensed documentation is available only to licensees of the product. A version of the Information Center that contains only unlicensed information is available through the publications ordering system, order number SK3T-6945.

Entitlement hardcopy books

The following essential publications, in hardcopy form, are provided automatically with the product. For more information, see “The entitlement set.”

The entitlement set

The entitlement set comprises the following hardcopy books, which are provided automatically when you order CICS Transaction Server for z/OS, Version 3 Release 1:

Memo to Licensees, GI10-2559
CICS Transaction Server for z/OS Program Directory, GI10-2586
CICS Transaction Server for z/OS Release Guide, GC34-6421
CICS Transaction Server for z/OS Installation Guide, GC34-6426
CICS Transaction Server for z/OS Licensed Program Specification, GC34-6608

You can order further copies of the following books in the entitlement set, using the order number quoted above:

CICS Transaction Server for z/OS Release Guide
CICS Transaction Server for z/OS Installation Guide
CICS Transaction Server for z/OS Licensed Program Specification

PDF-only books

The following books are available in the CICS Information Center as Adobe Portable Document Format (PDF) files:

CICS books for CICS Transaction Server for z/OS

General

CICS Transaction Server for z/OS Program Directory, GI10-2586
CICS Transaction Server for z/OS Release Guide, GC34-6421
CICS Transaction Server for z/OS Migration from CICS TS Version 2.3, GC34-6425

CICS Transaction Server for z/OS Migration from CICS TS Version 1.3,
GC34-6423

CICS Transaction Server for z/OS Migration from CICS TS Version 2.2,
GC34-6424

CICS Transaction Server for z/OS Installation Guide, GC34-6426

Administration

CICS System Definition Guide, SC34-6428

CICS Customization Guide, SC34-6429

CICS Resource Definition Guide, SC34-6430

CICS Operations and Utilities Guide, SC34-6431

CICS Supplied Transactions, SC34-6432

Programming

CICS Application Programming Guide, SC34-6433

CICS Application Programming Reference, SC34-6434

CICS System Programming Reference, SC34-6435

CICS Front End Programming Interface User's Guide, SC34-6436

CICS C++ OO Class Libraries, SC34-6437

CICS Distributed Transaction Programming Guide, SC34-6438

CICS Business Transaction Services, SC34-6439

Java Applications in CICS, SC34-6440

JCICS Class Reference, SC34-6001

Diagnosis

CICS Problem Determination Guide, SC34-6441

CICS Messages and Codes, GC34-6442

CICS Diagnosis Reference, GC34-6899

CICS Data Areas, GC34-6902

CICS Trace Entries, SC34-6443

CICS Supplementary Data Areas, GC34-6905

Communication

CICS Intercommunication Guide, SC34-6448

CICS External Interfaces Guide, SC34-6449

CICS Internet Guide, SC34-6450

Special topics

CICS Recovery and Restart Guide, SC34-6451

CICS Performance Guide, SC34-6452

CICS IMS Database Control Guide, SC34-6453

CICS RACF Security Guide, SC34-6454

CICS Shared Data Tables Guide, SC34-6455

CICS DB2 Guide, SC34-6457

CICS Debugging Tools Interfaces Reference, GC34-6908

CICSplex SM books for CICS Transaction Server for z/OS

General

CICSplex SM Concepts and Planning, SC34-6459

CICSplex SM User Interface Guide, SC34-6460

CICSplex SM Web User Interface Guide, SC34-6461

Administration and Management

CICSplex SM Administration, SC34-6462

CICSplex SM Operations Views Reference, SC34-6463

CICSplex SM Monitor Views Reference, SC34-6464

CICSplex SM Managing Workloads, SC34-6465

CICSplex SM Managing Resource Usage, SC34-6466

CICSplex SM Managing Business Applications, SC34-6467

Programming

CICSplex SM Application Programming Guide, SC34-6468

CICSplex SM Application Programming Reference, SC34-6469

Diagnosis

CICSplex SM Resource Tables Reference, SC34-6470
CICSplex SM Messages and Codes, GC34-6471
CICSplex SM Problem Determination, GC34-6472

CICS family books

Communication

CICS Family: Interproduct Communication, SC34-6473
CICS Family: Communicating from CICS on System/390, SC34-6474

Licensed publications

The following licensed publications are not included in the unlicensed version of the Information Center:

CICS Diagnosis Reference, GC34-6899
CICS Data Areas, GC34-6902
CICS Supplementary Data Areas, GC34-6905
CICS Debugging Tools Interfaces Reference, GC34-6908

Other CICS books

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 3 Release 1.

<i>Designing and Programming CICS Applications</i>	SR23-9692
<i>CICS Application Migration Aid Guide</i>	SC33-0768
<i>CICS Family: API Structure</i>	SC33-1007
<i>CICS Family: Client/Server Programming</i>	SC33-1435
<i>CICS Transaction Gateway for z/OS Administration</i>	SC34-5528
<i>CICS Family: General Information</i>	GC33-0155
<i>CICS 4.1 Sample Applications Guide</i>	SC33-1173
<i>CICS/ESA 3.3 XRF Guide</i>	SC33-0661

Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager® softcopy versions of a publication are usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

Subsequent updates will probably be available in softcopy before they are available in hardcopy. This means that at any time from the availability of a release, softcopy versions should be regarded as the most up-to-date.

For CICS Transaction Server books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Updates to the softcopy are clearly marked by revision codes (usually a # character) to the left of the changes.

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 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

Special characters

<apphandler>
 pipeline configuration element 125
<authentication>
 pipeline configuration element 138
<cics_soap_1.1_handler>
 pipeline configuration element 128
<cics_soap_1.2_handler>
 pipeline configuration element 130
<default_http_transport_handler_list>
 pipeline configuration element 132
<default_mq_transport_handler_list>
 pipeline configuration element 133
<default_target>
 pipeline configuration element 136
<default_transport_handler_list>
 pipeline configuration element 133
<dfhwsse_configuration>
 pipeline configuration element 137
<encrypt_body>
 pipeline configuration element 142
<handler>
 pipeline configuration element 134
<named_transport_entry>
 pipeline configuration element 125
<provider_pipeline>
 pipeline configuration element 125
<requester_pipeline>
 pipeline configuration element 127
<service_handler_list>
 pipeline configuration element 135
<service_parameter_list>
 pipeline configuration element 136
<service>
 pipeline configuration element 134
<sign_body>
 pipeline configuration element 141
<terminal_handler>
 pipeline configuration element 126
<transport_handler_list>
 pipeline configuration element 127
<transport>
 pipeline configuration element 136
<wsse_handler>
 pipeline configuration element 137

A

algorithm 182, 184
apphandler
 pipeline configuration element 125
assistant, Web services 53
atomic transaction 171, 177
 configuring CICS 173
 configuring service provider 174
 configuring service requester 175
 registration services 171

atomic transaction (*continued*)
 states 178
authentication
 pipeline configuration element 138

B

batch utility
 Web services assistant 53
body, SOAP 11

C

C and C++
 mapping to XML Schema 91
cics_soap_1.1_handler
 pipeline configuration element 128
cics_soap_1.2_handler
 pipeline configuration element 130
COBOL
 mapping to XML Schema 84
compliance with standards 33
configuration file, pipeline 119
configuring CICS 187
configuring RACF 185
container
 context container
 DFH-HANDLERPLIST 168
 DFH-SERVICEPLIST 168
 DFHWS-APPHANDLER 169
 DFHWS-DATA 169
 DFHWS-PIPELINE 169
 DFHWS-SOAPLEVEL 169
 DFHWS-TRANID 170
 DFHWS-URI 170
 DFHWS-USERID 170
 DFHWS-WEBSERVICE 170
 control container
 DFHERROR 160
 DFHFUNTION 161
 DFHHTTPSTATUS 163
 DFHMEDIATYPE 164
 DFHNORESPONSE 164
 DFHREQUEST 164
 DFHRESPONSE 165
 DFH-HANDLERPLIST 168
 DFH-SERVICEPLIST 168
 DFHERROR 160
 DFHFUNTION 161
 DFHHTTPSTATUS 163
 DFHMEDIATYPE 164
 DFHNORESPONSE 164
 DFHREQUEST 164
 DFHRESPONSE 165
 DFHWS-APPHANDLER 169
 DFHWS-DATA 169
 DFHWS-PIPELINE 169

- container (*continued*)
 - DFHWS-SOAPLEVEL 169
 - DFHWS-TRANID 170
 - DFHWS-URI 170
 - DFHWS-USERID 170
 - DFHWS-WEBSERVICE 170
- containers
 - used in a pipeline 159
- context containers 168
- control containers 159

D

- default_http_transport_handler_list
 - pipeline configuration element 132
- default_mq_transport_handler_list
 - pipeline configuration element 133
- default_target
 - pipeline configuration element 136
- default_transport_handler_list
 - pipeline configuration element 133
- DFH-HANDLERPLIST container 168
- DFH-SERVICEPLIST container 168
- DFHERROR container 160
- DFHFUNCTION container 161
- DFHHTTPSTATUS container 163
- DFHLS2WS
 - cataloged procedure 62
- DFHMEDIATYPE container 164
- DFHNORESPONSE container 164
- DFHREQUEST container 164
- DFHRESPONSE container 165
- DFHWS-APPHANDLER container 169
- DFHWS-DATA container 169
- DFHWS-PIPELINE container 169
- DFHWS-SOAPLEVEL container 169
- DFHWS-TRANID container 170
- DFHWS-URI container 170
- DFHWS-USERID container 170
- DFHWS-WEBSERVICE container 170
- DFHWS2LS
 - cataloged procedure 70
- dfhwsse_configuration
 - pipeline configuration element 137
- diagnosing problems
 - service requester 193
- diagram
 - syntax 79
- dynamic routing
 - in a service provider 158
 - in a terminal handler 158

E

- encrypt_body
 - pipeline configuration element 142
- envelope, SOAP 11

F

- fault, SOAP 11

H

- handler
 - pipeline configuration element 134
- header, SOAP 11
- high level language structure
 - converting to WSDL 62

L

- language structure
 - converting to WSDL 62
- limitations at run time 115

M

- mapping to C and C++ 91
- mapping to COBOL 84
- mapping to PL/I 97
- maxOccurs
 - in XML schema 104
- message handler
 - non-terminal 148, 149, 150
- minOccurs
 - in XML schema 104

N

- named_transport_entry
 - pipeline configuration element 125
- non-terminal message handler 148, 149, 150
- notation
 - syntax 79

P

- persistent message 51
- persistent message support 52
- pipeline configuration
 - Web Services Security 137
- pipeline configuration element
 - <apphandler> 125
 - <authentication> 138
 - <cics_soap_1.1_handler> 128
 - <cics_soap_1.2_handler> 130
 - <default_http_transport_handler_list> 132
 - <default_mq_transport_handler_list> 133
 - <default_target> 136
 - <default_transport_handler_list> 133
 - <dfhwsse_configuration> 137
 - <encrypt_body> 142
 - <handler> 134
 - <named_transport_entry> 125
 - <provider_pipeline> 125
 - <requester_pipeline> 127
 - <service_handler_list> 135
 - <service> 134
 - <sign_body> 141
 - <terminal_handler> 126
 - <transport_handler_list> 127
 - <transport> 136

- pipeline configuration element (*continued*)
 - <wsse_handler> 137
- pipeline configuration file 119
- pipeline definition
 - service requester 123
- PL/I
 - mapping to XML Schema 97
- provider_pipeline
 - pipeline configuration element 125

R

- requester_pipeline
 - element of pipeline definition 123
 - pipeline configuration element 127
- run time limitations 115

S

- security for Web services 179
- service
 - pipeline configuration element 134
- service parameter list
 - <service_parameter_list> 136
- service provider application
 - creating from a data structure 55
 - using atomic transactions 174
- service requester
 - diagnosing problems 193
 - pipeline definition 123
- service requester application
 - using atomic transactions 175
- service_handler_list
 - pipeline configuration element 135
- service_parameter_list
 - service parameter list 136
- sign_body
 - pipeline configuration element 141
- SOAP
 - body 11
 - envelope 11
 - fault 11
 - header 11
 - overview 11
 - overview of SOAP 11
- SOAP message
 - encrypting 183
 - example 11
 - signing 181
 - structure 11
- SOAP message path 17
- SOAP Message Security 31
- SOAP messages
 - validating against XML Schema 61
 - XML Schema
 - validating SOAP message 61
- specifications 29
- standards 29
 - compliance 33
- syntax notation 79

T

- terminal_handler
 - pipeline configuration element 126
- transport
 - pipeline configuration element 136
- transport_handler_list
 - pipeline configuration element 127

U

- URI
 - for MQ transport 50
- user containers 171
- UsernameToken Profile 1.0 32
- utility program
 - Web services assistant 53

V

- validating SOAP messages 61
- Variable arrays 104

W

- Web services assistant 53
 - creating a service provider application 55
- Web Services Security
 - pipeline configuration 137
- Web Services Security (WSS) 179, 185, 187
- Web Services Security: SOAP Message Security 31
- Web Services Security: UsernameToken Profile 1.0 32
- Web Services Security: X.509 Certificate Token Profile 1.0 32
- workload management
 - in a service provider 158
 - in a terminal handler 158
- WS-AT 171
- WSDL
 - and application data structure 26
 - converting to language structure 70
- WSS: SOAP Message Security 31
- WSS: UsernameToken Profile 1.0 32
- WSS: X.509 Certificate Token Profile 1.0 32
- wsse_handler
 - pipeline configuration element 137

X

- X.509 Certificate Token Profile 1.0 32
- XML Schema 84, 91, 97

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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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:

IBM United Kingdom Limited
User Technologies Department (MP095)
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.



Program Number: 5655-M15

SC34-6458-08



Spine information:



CICS Transaction Server for z/OS CICS Web Services Guide

Version 3
Release 1