CICS Transaction Server for z/OS
Version 5 Release 3

Web Services Guide

SC34-7452-00

CICS Transaction Server for z/OS
Version 5 Release 3

Web Services Guide

SC34-7452-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 647.

This edition applies to the IBM CICS Transaction Server for z/OS Version 5 Release 3 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

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 the following roles:

- 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.

Location of topics in the Knowledge Center

The topics in this publication can also be found in the CICS Transaction Server for z/OS Knowledge Center. The Knowledge Center uses content types to structure how the information is displayed.

The Knowledge Center content types are generally task-oriented, for example: upgrading, configuring, and installing. Other content types include reference, overview, and scenario or tutorial-based information. The following mapping shows the relationship between topics in this publication and the Knowledge Center content types, with links to the external Knowledge Center:

Table 1. Mapping of PDF topics to Knowledge Center content types. This table lists the relationship between topics in the PDF and topics in the content types in the Knowledge Center

Set of topics in this publication	Location in the Knowledge Center
• Chapter 1, "Web services," on page 1 • Chapter 5, "Getting started with SOAP web services," on page 45 • Chapter 6, "Creating the web services infrastructure," on page 51 • "Creating a SOAP web service" on page 51 • Chapter 13, "Support for securing web services," on page 569 • Chapter 15, "Troubleshooting SOAP web services," on page 593	• Product overview • Getting started • Configuring • Developing applications • Securing • Troubleshooting and support

Changes in CICS Transaction Server for z/OS, Version 5 Release 3

For information about changes that have been made in this release, please refer to *What's New* in the Knowledge Center, or the following publications:

- *CICS Transaction Server for z/OS What's New*
- *CICS Transaction Server for z/OS Upgrading to CICS TS Version 5.3*

Any technical changes that are made to the text after release are indicated by a vertical bar (|) to the left of each new or changed line of information.

Chapter 1. Web services

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.
- CICS supports the HTTP and IBM® WebSphere® MQ transport protocols.
- CICS Transaction Server for z/OS includes the CICS web services assistant, a set of utility programs that help you map WSDL and JSON service descriptions into high-level programming language data structures and vice versa. The utility programs support the following programming languages:

  COBOL

  PL/I

  C

  C++
- CICS support for SOAP web services conforms to open standards, including the following standards:

  SOAP 1.1 and 1.2

  HTTP 1.1

  WSDL 1.1 and 2.0

  JSON
- CICS support for SOAP web services ensures maximum interoperability with other web services implementations by conditionally or fully complying with many web services specifications, including the WS-I Basic Profile Version 1.1. The profile is a set of nonproprietary web services specifications, with clarifications and amendments to those specifications, which, taken together, promote interoperability between different implementations of web services.
- CICS support for SOAP web services includes support for web services pipelines that are Java™-based and non-Java-based. Java-based pipelines are processed using the T8 TCBs, and non-Java-based pipelines are processed using the L8 TCBs. This reduces the amount of QR TCB processing required to process the web service.
- CICS uses the IBM z/OS XML System Services (XMLSS) parser to parse SOAP envelopes. The XMLSS parser uses 64-bit (above-the-bar) storage in the CICS region, leaving more storage below the bar for user programs. The XMLSS parser also allows XML parsing to be offloaded to an zEnterprise® Application Assist Processor (zAAP). The zAAP-eligible proportion of the infrastructure for a web service is small, but if zAAP capacity is available then this can reduce the cost of hosting web services in CICS. For more information, see the IBM Redbooks® publication zSeries Application Assist Processor (zAAP) Implementation.

What is a web service?

A web service is a generic term for an interoperable machine-to-machine software function that is hosted at a network addressable location.

A web service has an interface, which hides the implementation details 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 independence encourages web service based applications to be loosely coupled, component-oriented, cross-technology implementations. Web services can be used alone or with other web services to carry out a complex aggregation or a business transaction.

CICS supports two distinct web service protocols, the SOAP and the JavaScript Object Notation (JSON) protocols. These two protocols have distinct characteristics and advantages.

A SOAP *web service* has an interface that is described in a machine-processable format that is called the Web Service Definition Language (WSDL) document. A SOAP web service is described by using a standard, formal XML notion that provides all of the details necessary to interact with the service, including message formats, transport protocols, and location. Tools can be used to process the WSDL, and produce client programs capable of communicating with the service by using the XML-based SOAP protocol. SOAP can be a verbose communication protocol, but it has the advantage of extensibility; more specifications exist to support Enterprise qualities of service such as distributed two-phase-commit support and sophisticated security options.

A JSON *web service* is less formally defined. The data format is described by using JSON schema notation, and it requires use of the HTTP transport protocol. JSON is a more convenient data representation format for typical mobile devices and JavaScript based applications. But it lacks the extensibility options of SOAP, so offers fewer options for Enterprise qualities of service. It is a lightweight protocol in contrast to SOAP.

Use JSON when you want to connect to CICS from mobile devices. Use SOAP when you want server to server communication.

Differences between SOAP and JSON web services in CICS

There are some important differences between SOAP and JSON:

- The content of a SOAP message is XML data, whereas a JSON message contains JSON data. JSON and XML are different encoding mechanisms for describing structured data.
- JSON tends to be a more efficient encoding mechanism, so a typical JSON message is smaller than the equivalent XML message.
- JSON is easy to integrate in JavaScript applications, but XML is not. This difference makes JSON a preferred data format for many mobile application developers.
- SOAP provides a mechanism to add Headers to a message, and a family of specifications for qualities of service (such as security configuration, and distributed transactions). JSON does not provide this mechanism. It instead relies on the services of the underlying HTTP network protocol. This reliance results in fewer options for securing and configuring a workload. The JSON architecture is often described as lightweight compared to SOAP.
- SOAP web services are described by using WSDL documents. JSON web services are structured less formally; they tend to be loosely coupled and prefer informal documentation often including examples.
- SOAP has a larger ecosystem of related tools that can help with application development.

- SOAP web services have an explicit error format that involves using SOAP Fault messages. There is no equivalent for JSON.
- SOAP web services support use of both HTTP and WebSphere MQ based messaging, JSON requires HTTP.
- JSON web services support both a RESTful and a Request-Response driven interface, SOAP supports the Request-Response interface only.
- SOAP web services support the **INVOKE** API command in CICS; using this API command requester (or client) mode applications can call remote SOAP web services. JSON does not support the **INVOKE** command, but JSON client applications can be hosted in CICS through use of the basic WEB API.

Despite these differences, there are also many similarities between JSON and SOAP. Both protocols are cross-vendor open technologies, and both share infrastructure in CICS, the same CICS supplied tools, and much of the same configuration. It is possible to provide both SOAP and JSON interfaces to an existing application program hosted in CICS.

For more information about SOAP, see Getting started with SOAP web services in Getting started. For more information about JSON, see Getting started with JSON web services in Getting started.

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. Use web services to integrate your applications into the Web.

Web services can help your business in these ways:

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

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

With the support that CICS provides for web services, you can deploy your existing applications in new ways, with the minimum amount of reprogramming.

Web services terminology

You must be familiar with these terms to understand the topics in the web services section.

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).

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

JavaScript Object Notation (JSON)
A lightweight data-interchange format that is based on the object-literal

notation of JavaScript. JSON is programming-language neutral but uses conventions from languages that include C, C++, C#, Java, JavaScript, Perl, Python.

JSON schema
A JavaScript Object Notation document that describes the structure and constrains the contents of other JSON documents.

RESTful
Pertaining to applications and services that conform to Representational State Transfer (REST) constraints.

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.

The specification for SOAP 1.2 is published here:

SOAP Version 1.2 Part 0: Primer

SOAP Version 1.2 Part 1: Messaging Framework

SOAP Version 1.2 Part 2: Adjuncts

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 forwards a SOAP message toward an ultimate SOAP receiver.

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

SOAP node
Processing logic that 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 See 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 that they offer, so that other businesses can find them. The specification is published by OASIS.

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 Addressing
Web Services Addressing (WS-Addressing) provides a transport-neutral mechanism to address web services and messages.

The specifications for WS-Addressing are published here:

- Web Services Addressing 1.0 - Core
- Web Services Addressing 1.0 - SOAP Binding
- Web Services Addressing 1.0 - Metadata
- Web Services Addressing- Submission

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 by OASIS at Web Services Atomic Transaction.

Web service binding file
A file, associated with a WEBSERVICE resource, that 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 function is offered.

The specification is published at Web Services Description Language (WSDL).

Web Services Security
A set of enhancements to SOAP messaging that provides message integrity and confidentiality. The specification is published by OASIS at Web Services Security: SOAP Message Security 1.0 (WS-Security 2004).

WS-Atomic Transaction
See Web Services Atomic Transaction.

WS-I Basic Profile
A set of nonproprietary web services specifications, 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.

WSDL
See Web Service Description Language.

WSS See Web Services Security.

XML Extensible Markup Language.

The specifications for XML are published here:

SOAP Version 1.2 Part 0: Primer

SOAP Version 1.2 Part 1: Messaging Framework

SOAP Version 1.2 Part 2: Adjuncts

XML namespace
A collection of names, identified by a URI reference, that 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. SOAP web services architecture

The SOAP web services architecture is based on 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.

- 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.

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

The service registry
The service registry is a central location where service providers can publish their service descriptions and where service requesters can find those service descriptions.

The registry is an optional component of the web services architecture because service requesters and providers can communicate without it in many situations. For example, the organization that provides a service can distribute the service description directly to the users of the service in a number of ways, including offering the service as a download from an FTP site.

Using a service registry offers a number of advantages to both the requester and provider; for example, using the IBM WebSphere Service Registry and Repository (WSRR) can help the requester to find services more quickly and can help the provider to enforce version control of the services being offered.

CICS provides direct support for implementing service requester and service provider components. However, you need additional software to deploy a service registry in CICS. If you use the IBM WebSphere Service Registry and Repository (WSRR), CICS provides support for WSRR through the web services assistant. Alternatively, you can deploy a service registry on another platform.

Interactions between a service provider, a service requester, and, a service registry

The interactions between the service provider, service requester, and service registry involve the following operations:

Publish
When a service registry is used, a service provider publishes its service description in a service registry for the service requester to find.

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

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

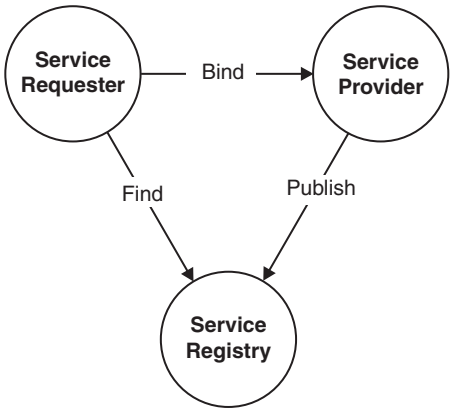


Figure 1. web services components and interactions

Web service description

A web service description is a document by which the *service provider* communicates the specifications for starting 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.

A service description can conform to either the WSDL 1.1 or WSDL 2.0 specification. Each has differences in both the terminology and major elements that can be included in the service description. The following information uses WSDL 1.1 terminology and elements to explain the purpose of the service description.

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

- An abstract *service interface definition* that describes the interfaces of the service and makes it possible to write programs that implement and start the service.
- A concrete *service implementation definition* that describes the location on the network (or *endpoint*) of the web service of the provider and other implementation-specific details. It enables a service requester to connect to the service provider.

See Figure 2 on page 9.

A WSDL 1.1 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.

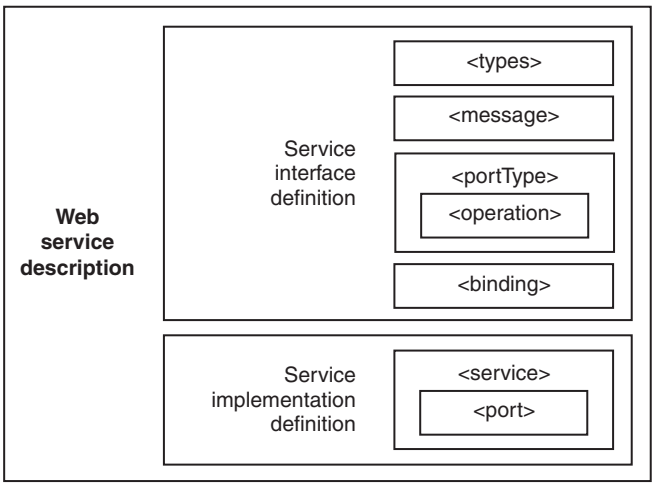


Figure 2. Structure of a web service description

Because you can partition the web service description, you can divide responsibility for creating a complete service description. As an illustration, consider a service that is defined by a standards body for use across an industry and is implemented by individual companies in that industry:

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

  <types>

  <message>

<portType>

<binding

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

  <port>

  <service>

Service publication

You can publish a service description using a number of different mechanisms. Each mechanism is suitable for use in different situations. CICS supports the use of the IBM WebSphere Service Registry and Repository (WSRR) for publishing service descriptions. Alternatively, you can use other methods to publish a service description.

WSSR CICS supports the use of WSRR for publishing service descriptions. For more information about the support that CICS provides for WSSR, see the "Interoperability between the web services assistant and WSRR" topic in the Information Center.

Any of the following mechanisms, none of which is directly supported by CICS, can be used with CICS to publish service descriptions:

Direct publishing
This mechanism is the most straightforward for publishing service descriptions; the service provider sends the service description directly to the service requester, using an e-mail attachment, an FTP site, or a CD ROM distribution.

DISCO
These proprietary protocols provide a dynamic publication mechanism. The service requester uses a simple HTTP GET mechanism to retrieve a web service description 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 that they offer so that other businesses can find them.

A service description can be published in more than one form if required.

Chapter 3. 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 various 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. Consult these documents for 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 might 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 includes this information:

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

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 in the <Body>, is used for reporting errors.

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

The SOAP header
The SOAP <Header> is an optional subelement of the SOAP envelope. It 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 subelement of the SOAP envelope. It contains information intended for the ultimate recipient of the message.

The SOAP fault
The SOAP <Fault> is a subelement 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 in the <Header> and the <Body> are defined by the applications that make use of them. However, 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 blocks

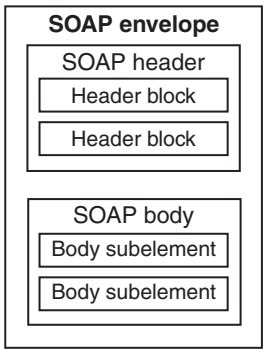


Figure 3. The structure of a SOAP message

(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 in 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 It represents a logical grouping of data that 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 processes 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 what will 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. In this way, a SOAP message can 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. Note these important attributes:

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 operates 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 has 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 that performs logging. Header blocks that 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 the ones that are defined by the application:

http://www.w3.org/2003/05/soap-envelope/none
None of the SOAP nodes on the message path will 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 that are important to the overall purpose of the application. If a SOAP node determines, using the **role** or **actor** attribute, that it will process a header block, and the **mustUnderstand** attribute has a value of `"true"`, 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.

The **mustUnderstand** attribute has these values:

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 in 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, which is used for reporting errors. Other elements in the <Body> are defined by the web service that uses them.

The SOAP fault

The SOAP `<Fault>` element carries error and status information in the SOAP message.

If an error occurs in a web service, a fault message is returned to the client. The basic structure of the fault message is defined in the SOAP specifications. Each fault message can include XML that describes the specific error condition. For example, if an application abend occurs in a CICS web service, a fault message is returned to the client reporting the abend.

CICS can send different types of fault message:

- Standard SOAP fault messages are defined by the SOAP specifications or one of the web service specifications that are supported in CICS. The faults report common error conditions, such as malformed SOAP envelopes.
- Application SOAP fault messages are generated using the **EXEC CICS SOAPFAULT** API commands in response to conditions that are detected or handled by the application. The structure of these fault messages is known to the application, but not to CICS.
- SOAP handler fault messages are generated by the SOAP handler programs in response to general error handling in CICS. For example, the SOAP handler programs send SOAP faults for abends, XML parsing failures, and other common errors.
- Application handler fault messages are generated by CICS SOAP application handlers in response to finding errors when processing the body of a SOAP message. These faults occur during the process of transforming the XML into binary application data or when generating the response.

If an error occurs, the SOAP `<Fault>` element must be a body entry and must not be present more than once in a `<Body>` element. The XML elements inside 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 elements:

<faultcode>
The <faultcode> element is a mandatory element in 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 in 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 were not 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 in header entries.

SOAP 1.2

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

<Code>
The <Code> element is a mandatory element in 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 in the <Fault> element. The <Reason> element contains one or more <Text> elements, each of which contains information about the fault in a different native 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 in which the node was operating 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 regarding which parts of the faulty SOAP message were processed.

SOAP fault example and schemas

The following example shows a SOAP fault message that is generated by the application handler, DFHPITP, when processing the body of a SOAP message.

```
<SOAP-ENV:Fault xmlns="">
   <faultcode>SOAP-ENV:Server</faultcode>
   <faultstring>Conversion to SOAP failed</faultstring>
      <detail>
         <CICSFault xmlns="http://www.ibm.com/software/htp/cics/WSFault">
          DFHPI1008 25/01/2010 14:16:50 IYCWZCFU 00340 XML
          generation failed because of incorrect input
          (CONTAINER_NOT_FOUND container name) for WEBSERVICE
          servicename.
          </CICSFault>
      </detail>
</SOAP-ENV:Fault>
```

Most of the content in this example is common to all fault messages. The `<Detail>` element contains the unique information that describes the problem that was encountered by the application handler. This specific fault message contains a copy of an error message that is written to the CICS message logs. If you want to parse application handler SOAP faults programmatically, use the following XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com/software/htp/cics/WSFault"
xmlns:tns="http://www.ibm.com/software/htp/cics/WSFault"
elementFormDefault="qualified">
   <element name="CICSFault" type="string">
      <annotation>
         <documentation>
            The value of this element is a text string that describes a
            problem encountered during the processing of the Body of a
            SOAP message.
         </documentation>
      </annotation>
   </element>
</schema>
```

The general purpose fault messages are more complicated because the `<Detail>` section can be structured in several different ways. If you want to parse SOAP handler faults programmatically, use the XML schema that is supplied in *usshome*`/schemas/soapfault/soapfault.xsd`, where *usshome* is the value of the **USSHOME** system initialization parameter.

SOAP nodes

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

A SOAP node can perform these operations:

- Transmit a SOAP message
- Receive a SOAP message
- Process a SOAP message
- Relay a SOAP message

A SOAP node can be one of these types:

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, targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message toward an ultimate SOAP receiver.

Ultimate SOAP receiver
The SOAP receiver that is a final destination of a SOAP message. It processes 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, including 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.

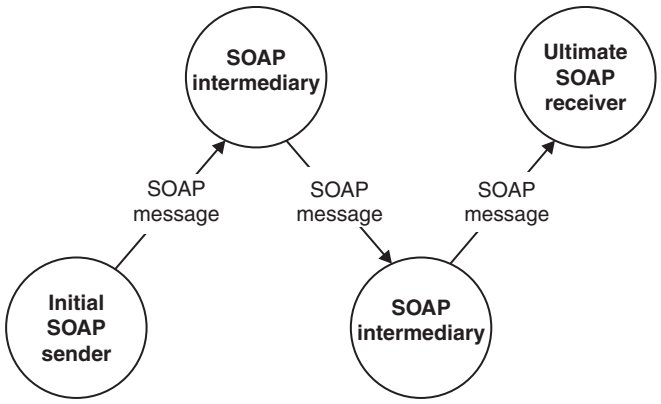


Figure 5. An example of a SOAP message path

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 toward its ultimate receiver.

The *ultimate SOAP receiver* is the final destination of a SOAP message. As well as processing the header blocks, it processes 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 SOAP web services

CICS supports two different approaches to the deployment of 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 that operate on web service requests and responses.

When you deploy your CICS applications in a web services environment you can choose from the following options:

- 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.
- 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 that 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:

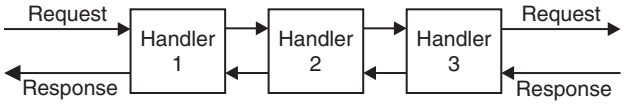


Figure 6. A generic CICS pipeline

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 want 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 TCPIPSERVICE for the HTTP transport, a QUEUE for the MQ transport) is in use.

This is illustrated in Figure 7:

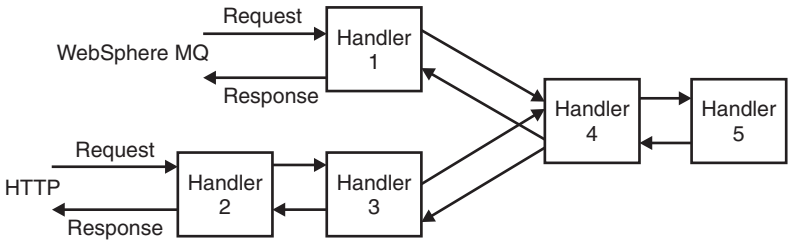


Figure 7. Pipeline with transport-related handlers

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.

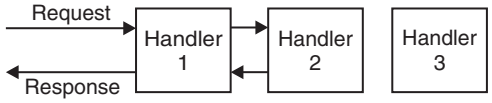


Figure 8. Interrupting the pipeline flow

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:

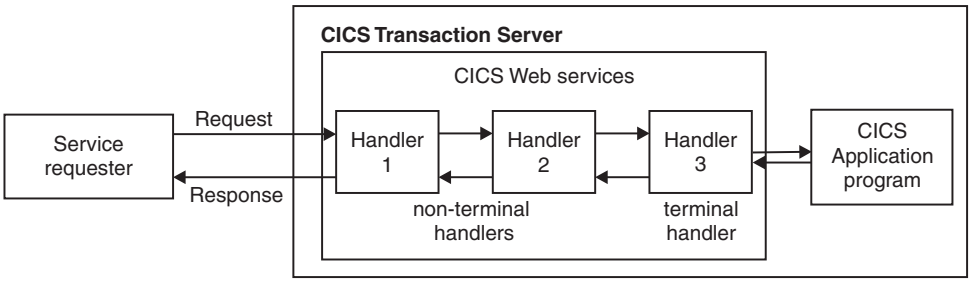


Figure 9. A service provider pipeline

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:

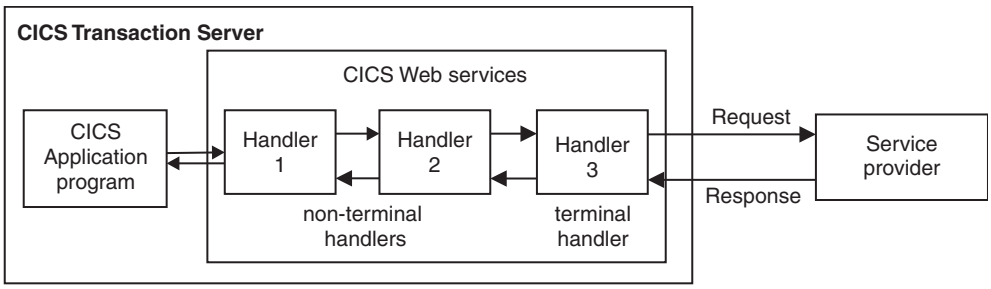


Figure 10. A service requester pipeline

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.

- 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.

- A service requester pipeline can be configured to support SOAP 1.1 or SOAP 1.2, but not both. However, a service provider pipeline can be configured to support both SOAP 1.1 and SOAP 1.2. Within your CICS system, you can have many pipelines, some of which support SOAP 1.1 or SOAP 1.2 and some of which support both.
- 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 Version 1.1, 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 on page 27 shows where these files are used in a service provider.

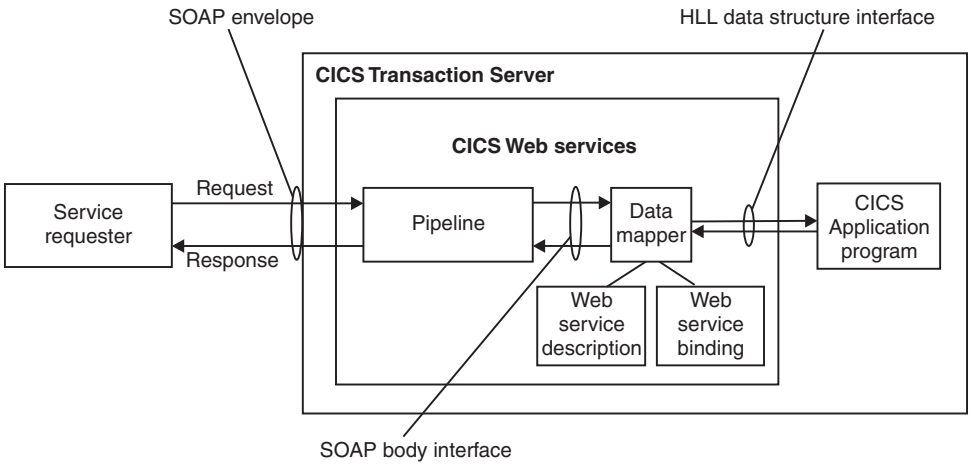


Figure 11. Mapping the SOAP body to the application data structure in a service provider

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.

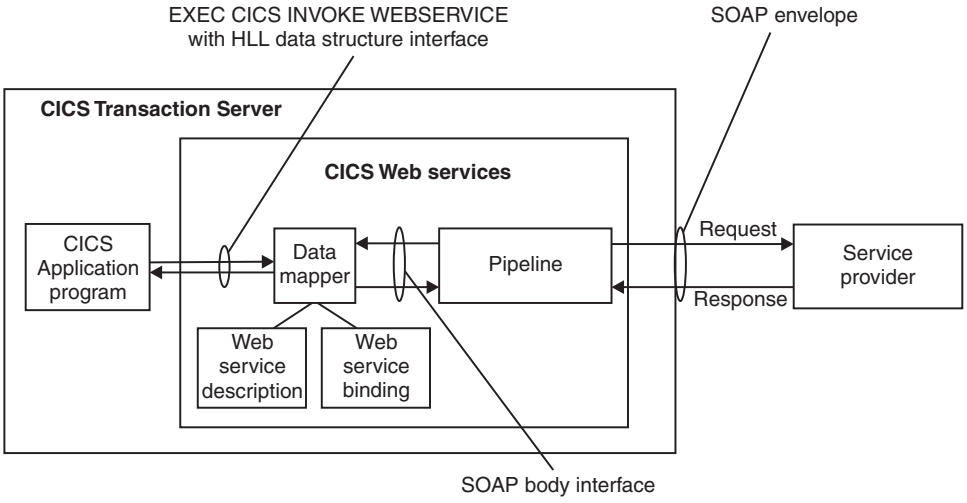


Figure 12. Mapping the SOAP body to the application data structure in a service requester

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 30:

```
<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

WSDL and message exchange patterns

A WSDL 2.0 document contains a message exchange pattern (MEP) that defines the way that SOAP 1.2 messages are exchanged between the web service requester and web service provider.

CICS supports four out of the eight message exchange patterns that are defined in the *WSDL 2.0 Part 2: Adjuncts* specification and the *WSDL 2.0 Part 2: Additional MEPs* specification for both service provider and service requester applications. The following MEPs are supported:

In-Only

A request message is sent to the web service provider, but the provider is not allowed to send any type of response to the web service requester.

- In provider mode, when CICS receives a request message from a web service that uses the In-Only MEP, it does not return a response message. The DFHNORESPONSE container is put in the SOAP handler channel to indicate that the pipeline must not send a response message.

- In requester mode, CICS sends the request message to the web service provider and does not wait for a response.

In-Out

A request message is sent to the web service provider, and a response message is returned to the web service requester. The response message could be a normal SOAP message or a SOAP fault.

- In provider mode, when CICS receives a request message from a web service that uses the In-Out MEP, it returns a response message to the requester.
- In requester mode, CICS sends a request message and waits for a response. This response is either a normal response message or a SOAP fault message. The length of time that CICS waits for a response is configured in the pipeline and applies to all web services using that pipeline. If the request times out before CICS receives a response, an error is returned to the service requester application.

In-Optional-Out

A request message is sent to the web service provider, and a response message is optionally returned to the web service requester. If there is a response, it could be either a normal SOAP message or a SOAP fault.

- In provider mode, the decision about whether to return a SOAP response message, a SOAP fault, or no response, happens at run time and is dependant on the service provider application logic. If CICS does not send a response to the web service requester, the DFHNORESPONSE container is put in the SOAP handler channel to indicate that the pipeline must not send a response message. If no message is sent, the service provider application must delete the DFHWS-DATA container from the channel.
- In requester mode, CICS sends a request message and waits for a response from the web service requester. If the request times out before a response is received, CICS assumes that the message was received successfully and that the provider did not need to send a response. The length of time that CICS waits for a response is configured in the pipeline and applies to all web services using that pipeline.

Robust In-Only

A request message is sent to the web service provider, and a response message is only returned to the web service requester if an error occurs. If there is an error, a SOAP fault message is sent to the requester.

- In provider mode, if the pipeline successfully passes the request message to the application, a DFHNORESPONSE container is put in the SOAP handler channel to indicate that the pipeline must not send a response message. If an error occurs in the pipeline, a SOAP fault message is returned to the requester.
- In requester mode, CICS sends the request message to the web service provider and waits for a specified period before timing out. The length of time that CICS waits for a response is configured in the pipeline and applies to all web services using that pipeline. If there is a timeout, CICS assumes that the request message was received successfully.

For more information on message exchange patterns in WSDL 2.0, see the following W3C specifications:

- *WSDL 2.0 Part 2: Adjuncts*: .
- *WSDL 2.0 Part 2: Additional MEPs*: .

Related concepts:

"Message exchanges" on page 359
Web Services Addressing (WS-Addressing) supports these message exchanges: one-way, two-way request-response, synchronous request-response, and asynchronous request-response.

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 WEBSERVICE resource.

Related information:

WEBSERVICE resource definitions

External standards

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

SAML

SAML is an XML-based framework for describing and exchanging security information.

Security Assertion Markup Language (SAML) is an XML-based framework for describing and exchanging security information between online business partners. This security information is expressed in the form of portable SAML assertions that applications working across security domain boundaries can trust. The OASIS SAML standard defines precise syntax and rules for requesting, creating, communicating, and using these SAML assertions.

SAML provides a solution for a number of problems:

- It provides an open standard for exchanging security information between Service Providers, also known as Federated Identity.
- It provides a means for end-to-end auditing.
- It provides a common source for user role or authority-based information.

CICS supports the SAMLCore1.1 and SAMLCore2.0 standards. It does not support the protocols that are described in those standards. It does not support SAMLCore 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/SOAP. This specification has not been endorsed by the W3C, but forms the basis for the SOAP 1.2 specification.

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

- Part 1: Messaging Framework is published at http://www.w3.org/TR/soap12-part1/.
- 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 1.1 Binding for MTOM 1.0

SOAP 1.1 Binding for MTOM 1.0 is a specification that describes how to use the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications with SOAP 1.1.

The aim of this specification is to define the minimum changes to MTOM and XOP to enable these facilities to be used interoperably with SOAP 1.1 and to largely reuse the SOAP 1.2 MTOM/XOP implementation.

The SOAP 1.1 Binding for MTOM 1.0 specification is published as a formal submission by the World Wide Web Consortium (W3C) at http://www.w3.org/Submission/soap11mtom10/.

SOAP Message Transmission Optimization Mechanism (MTOM)

SOAP Message Transmission Optimization Mechanism (MTOM) is one of a related pair of specifications that defines conceptually how to optimize the transmission and format of a SOAP message.

MTOM defines:

1. how to optimize the transmission of base64binary data in SOAP messages in abstract terms

2. how to implement optimized MIME multipart serialization of SOAP messages in a binding independent way using XOP

The implementation of MTOM relies on the related XML-binary Optimized Packaging (XOP) specification. As these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

The specification is published by the World Wide Web Consortium (W3C) as a W3C Recommendation at http://www.w3.org/TR/soap12-mtom/.

Web Services Addressing 1.0

Web Services Addressing 1.0 (WS-Addressing) is a specification that defines a transport-independent mechanism for passing messaging information between web services.

The WS-Addressing specification defines two constructs, message addressing properties and endpoint references, that normalize the information that is typically provided by transport protocols and messaging systems.

The specification is published by the World Wide Web Consortium (W3C) as a W3C recommendation and is published in three parts:

- WS-Addressing 1.0 - Core
- WS-Addressing 1.0 - SOAP binding
- WS-Addressing 1.0 - Metadata

You are recommended to follow these W3C specifications when using WS-Addressing with CICS.

For interoperability, CICS tolerates the W3C WS-Addressing submission specification only when the namespace is set to: `http://schemas.xmlsoap.org/ws/2004/08/addressing`.

The CICS API commands support MAPs and EPRs that follow the WS-Addressing recommendation specifications; however, the API commands do not support MAPs and EPRs that follow the WS-Addressing submission specification.

The addressing context maintains all the MAPs at the level of the recommendation specifications. If required, these MAPs can be converted to, or from, the submission specification level when they are applied to, or extracted from, the SOAP message.

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 http://www.ibm.com/developerworks/library/specification/ws-tx/.

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 http://www.ibm.com/developerworks/library/specification/ws-tx/.

Web Services Description Language Version 1.1 and 2.0

Web Services Description Language (WSDL) 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.

WSDL 2.0 provides a model as well as an XML format for describing web services. It enables you to separate the description of the abstract functionality offered by a service from the concrete details of a service description, such as "how" and "where" that functionality is offered. It also describes extensions for Message Exchange Patterns, SOAP modules, and a language for describing such concrete details for SOAP 1.2 and HTTP. The WSDL 2.0 specification also resolves many technical issues and limitations that are present in WSDL 1.1.

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

The specification for WSDL 2.0 is published as a W3C recommendation at http://www.w3.org/TR/wsdl20.

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

Web Services Trust Language

Web Services Trust Language (or WS-Trust) defines extensions that build on Web Services Security to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

WS-Trust describes:

1. Methods for issuing, renewing, and validating security tokens.
2. Ways to establish, access the presence of, and broker trust relationships.

CICS supports the April 2012 version of the specification which is published at OASIS WS-Trust v1.4 Standard.

WSDL 1.1 Binding Extension for SOAP 1.2

WSDL 1.1 Binding Extension for SOAP 1.2 is a specification that defines the binding extensions that are required to indicate that web service messages are bound to the SOAP 1.2 protocol.

The aim of this specification is to provide functionality that is comparable with the binding for SOAP 1.1.

This specification is published as a formal submission request by the World Wide Web Consortium (W3C) at http://www.w3.org/Submission/wsdl11soap12/.

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 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.

XML (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-binary Optimized Packaging (XOP)

XML-binary Optimized Packaging (XOP) is one of a related pair of specifications that defines how to efficiently serialize XML Infosets that have certain types of content.

XOP does this by:

1. packaging the XML in some format. This is called the *XOP package*. The specification mentions MIME Multipart/Related but does not limit it to this format.
2. Re-encoding all or part of base64binary content to reduce its size.
3. Placing the base64binary content elsewhere in the package and replacing the encoded content with XML that references it.

XOP is used as an implementation of the MTOM specification, which defines the optimization of SOAP messages. As these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

The specification is published by the World Wide Web Consortium (W3C) as a W3C Recommendation at http://www.w3.org/TR/xop10/

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 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 http://www.w3.org/TR/xmldsig-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-Addressing

CICS complies with the Core and SOAP binding parts of the WS-Addressing specification. CICS complies with the Metadata part of the specification with one exception.

When CICS issues a WS-Addressing fault, it does not conform to the specification. CICS follows the format described in the Metadata specification for the default action when building a WS-Addressing fault, but it does not include the final delimiter and Fault name.

For WSDL 1.1, the default action according to the specification is:

```
[target namespace][delimiter][port type name][delimiter][operation name][delimiter]Fault[delimiter][fault name]
```

However, CICS omits the fault name and builds the default action as follows:

```
[target namespace][delimiter][port type name][delimiter][operation name][delimiter]Fault[delimiter]
```

For WSDL 2.0, the default action according to the specification is:

```
[target namespace][delimiter][interface name][delimiter][fault name]
```

However, CICS omits the fault name and builds the default action as follows:

```
[target namespace][delimiter][interface name][delimiter]
```

How CICS complies with WSDL 2.0

CICS conditionally complies with WSDL 2.0, and support is subject to the following restrictions.

Mandatory requirements

- Only the message exchange patterns in-only, in-out, robust in-only, and in-optional-out may be used in the WSDL.
- Only one Endpoint is allowed for each Service.
- There must be at least one Operation.
- Endpoints may only be specified with a URI.
- There must be a SOAP binding
- The XML schema type system must be used.

Aspects that are tolerated

- The following HTTP binding properties are ignored:
  - whttp:location
  - whttp:header
  - whttp:transferCodingDefault
  - whttp:transferCoding
  - whttp:cookies
  - whttp:authenticationType
  - whttp:authenticationRealm
- SOAP header information is ignored by DFHWS2LS. However, you can add your own message handlers to the pipeline to create and process the required SOAP header information for inbound and outbound messages.

Aspects that are not supported

- The #any and #other message content models.
- The out-only, robust-out-only, out-in and out-optional-in message exchange patterns.
- WS-Addressing for Endpoints.
- HTTP GET is not supported. This is defined using the soap-response message exchange pattern in the WSDL document. If your WSDL defines this message exchange pattern, DFHWS2LS issues an error message.

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)
- SAML token
- Kerberos token

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 `<wsse: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 algorithm requirements as those specified in the XML Signature specification. CICS supports:

Algorithm type	Algorithm	URI
Digest	SHA1	`http://www.w3.org/2000/09/xmldsig#sha1`
Signature	DSA with SHA1 (validation only)	`http://www.w3.org/2000/09/xmldsig#dsa-sha1`
Signature	RSA with SHA1	`http://www.w3.org/2000/09/xmldsig#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`
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.pdf.
- 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.pdf.
- X.509 Version 3: PKCS7 with or without CRLs. The IBM Software Development Kit (SDK) supports both.

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

- References to X509 certificates from a <wsse:SecurityTokenReference> using a <wsse:KeyIdentifier>
- 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 algorithm for encryption, which is optional in the XML encryption specification, is not supported:
  - Canonical XML with or without comments
  - Exclusive XML canonicalization with or without comments
- In the Username Token Version 1.0 Profile specification, the digest password type is not supported.

How CICS complies with WS-Trust

CICS conditionally complies with WS-Trust, and support is subject to the following restrictions.

Aspects that are supported

- Validation binding
- Issuance binding where one token is returned
- AppliesTo in the Issuance binding

Aspects that are tolerated

- Requested references
- Keys and entropy
- Returning computed keys

Aspects that are not supported

- Returning multiple security tokens
- Returning security tokens in headers
- Renewal bindings
- Cancel bindings
- Negotiation and challenge extensions
- Key and Token parameter extensions
- Key exchange token binding

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 for a list of unsupported schema elements.

Conformance targets identify what artifacts (e.g. SOAP message, WSDL description) or parties (e.g. SOAP processor, 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.

Chapter 5. Getting started with SOAP web services

There are several ways to get started with SOAP web services in CICS. The most appropriate way for you depends on how much you already know and how advanced your plans are for using SOAP web services.

About this task

Here are some starting points for SOAP web services in CICS:

Procedure

- 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 Chapter 16, "The CICS catalog manager example application," on page 605
- Plan for the deployment of an application as a service provider or a requester. You might already know enough about how you will use web services in CICS to start planning your applications and the related infrastructure.

Planning to use SOAP web services

Before you can plan to use SOAP web services in CICS, you need to consider these questions for each application.

Before you begin

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 might 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 want to use other message formats. For example, you might have developed your own message formats that you want to deploy with the CICS web services infrastructure. You can do this with CICS, 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.

If you decide not to use SOAP, your application programs will be responsible for parsing inbound messages, and constructing outbound messages.

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.

Although the CICS web services assistant supports the most common data types and structures, there are some that are not supported. In this situation, you should check the list of unsupported data types and structures for the language in question, and consider providing a program layer that maps your application data to a format that the assistant can support. If this is not possible, you will need to parse the message yourself. For details on what the assistant can and cannot support, see High-level language and XML schema mapping.

If you decide not to use the CICS web services assistant, you can use a tool such as Rational® Developer for z System z® to create the necessary artifacts, and you can then provide your own code for parsing inbound messages, and constructing outbound messages. You can also use the provided vendor interface API.

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.

What to do next

Related information:

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

Planning a SOAP 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.

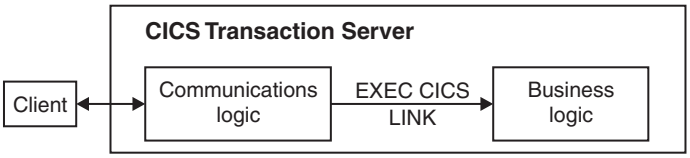


Figure 15. Application partitioned into communications and business logic

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

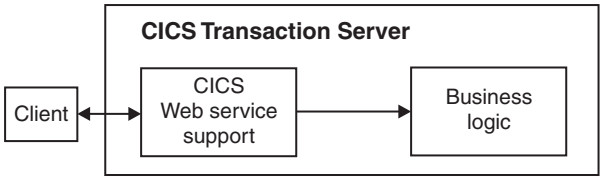


Figure 16. Simple deployment of CICS application as a web service provider

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:

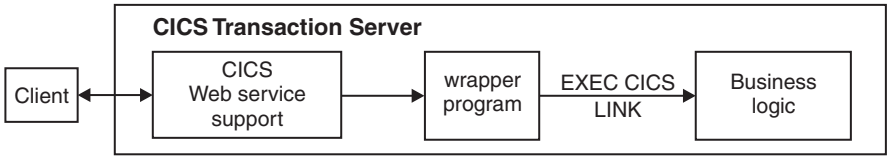


Figure 17. Deployment of CICS application as a web service provider using a wrapper program

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 SOAP 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 on page 49 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.

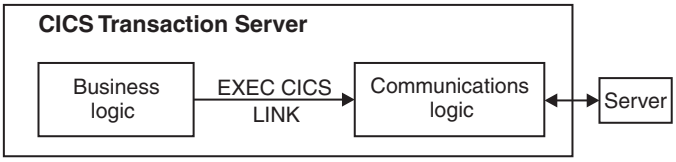


Figure 18. Application partitioned into communications and business logic

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 SERVICE** 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.

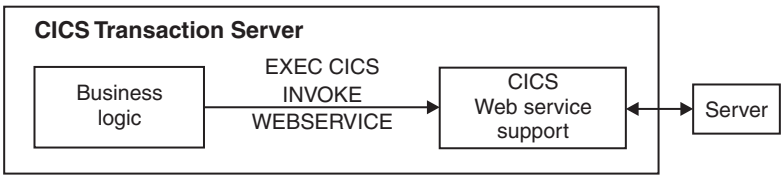


Figure 19. Simple deployment of CICS application as a web service requester

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 SERVICE** 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 on page 50.

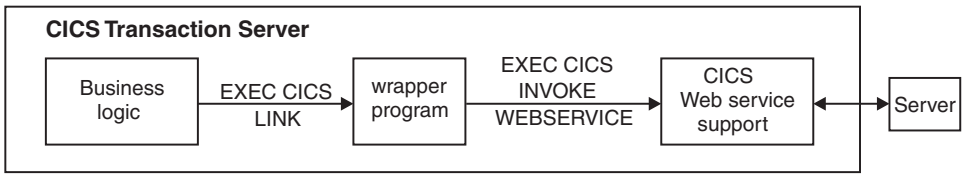


Figure 20. Deployment of CICS application as a web service requester using a wrapper program

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.

Chapter 6. Creating the web services infrastructure

To deploy a web service to CICS, you must create the necessary transport infrastructure and define one or more pipelines that will process your web services requests. Typically, one pipeline can process requests for many different web services, and, when you deploy a new web service in your CICS system, you can choose to use an existing pipeline.

Configuring your CICS system for web services

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

Procedure

1. Ensure that you have installed Language Environment® support for PL/I. For more information, see Installing Language Environment support.
2. Activate z/OS Support for Unicode. You must enable the z/OS conversion services and install a conversion image that specifies the data conversions that you want CICS to perform between SOAP messages and an application program. For more information, see z/OS Unicode Services User's Guide and Reference.

Creating a SOAP web service

You can expose existing CICS applications as SOAP web services and create new CICS applications to act as SOAP web service providers or requesters.

Before you begin

Before you begin to create a SOAP web service, perform these tasks:

1. Configure your CICS system to support web services; see "Configuring your CICS system for web services."
2. Create the necessary infrastructure to support the deployment of your web services; see Chapter 6, "Creating the web services infrastructure."
3. Decide whether you want to use the web services assistant; see "Planning to use SOAP web services" on page 45.

About this task

The CICS web services assistant is a supplied utility that helps you to create the necessary artifacts for a new SOAP web service provider or a service requester application, or to enable an existing application as a web service provider.

The CICS web services assistant can create a WSDL document from a simple language structure or a language structure from an existing WSDL document; it supports COBOL, C/C++, and PL/I. It also generates information that is used to enable automatic runtime conversion of the SOAP messages to containers and COMMAREAs, and vice versa. This information is used by the CICS web services support during pipeline processing.

Create your web service, as described in the following procedure, and validate that it works correctly:

Procedure

1. Create a SOAP web service in one of four ways:
   - Use the web services assistant to create the web service description or language structures and deploy them into CICS. Use the **PIPELINE SCAN** command to automatically create the required CICS resources.
   - Use Rational Developer for z Systems or the Java API to create the web service description or language structures and deploy them into CICS. Use the **PIPELINE SCAN** command to automatically create the required CICS resources.
   - Create or change an application program to handle the XML in the inbound and outbound messages, including the data conversion, and populate the correct containers in the pipeline. You must create the required CICS resources manually.
   - Deploy an Axis2 application as a web service.
2. Start the web service to test that it works as you intended. If you are using the web services assistant to deploy your web service, you can use the **SET WEBSERVICE** command to turn on validation. This validation checks that the data is converted correctly.

What to do next

These steps are explained in more detail in the following topics.

The CICS web services assistant

The CICS web services assistant is a set of batch utilities that 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.

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.

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.

For more information on the web services assistant's utility programs and data mappings, see the following topics.

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.

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 Starts the job.

EXEC Specifies the procedure name (DFHLS2WS).

INPUT.SYSUT1 DD
Specifies the input. The input parameters are typically 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 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/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 z/OS UNIX directory path used on other parameters. The default is the empty string.

Typically, you do not 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 z/OS UNIX 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 file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*.

Typically, you do not 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 at run time:

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 as follows:

/tmp/LS2WS.in

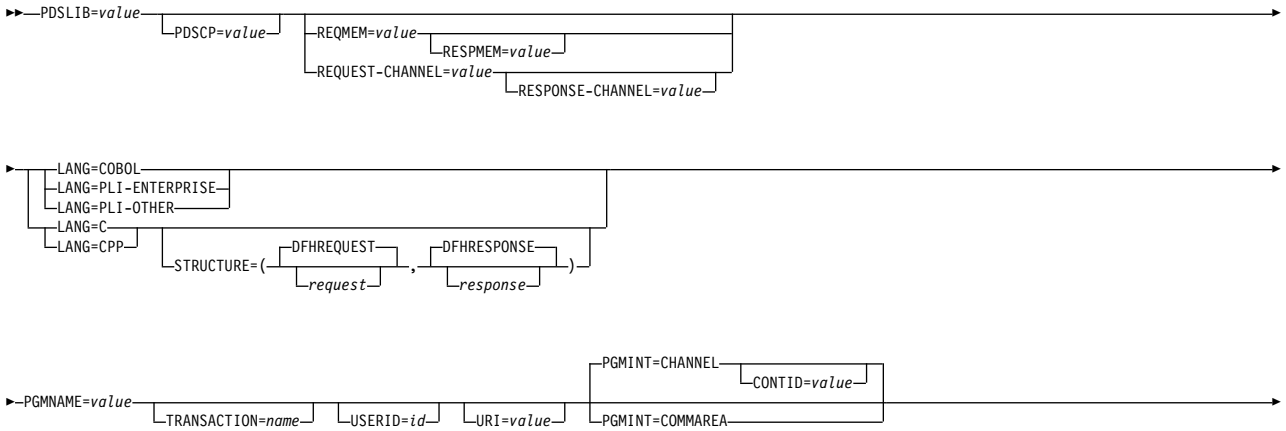
/tmp/LS2WS.out

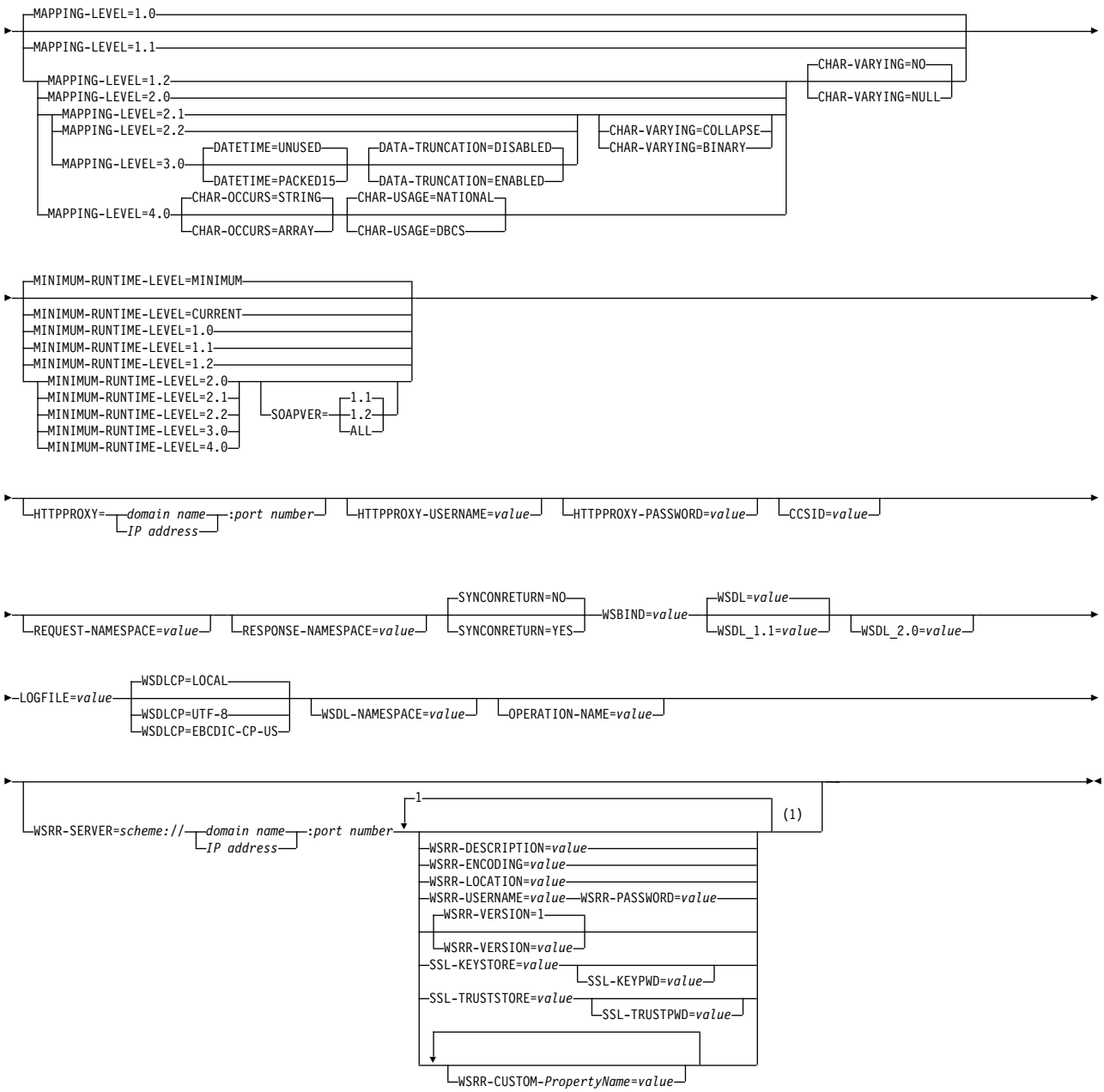
/tmp/LS2WS.err

Important: DFHLS2WS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHLS2WS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that 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





Notes:

1 Each of the WSRR parameters that can be specified when the **WSRR-SERVER** parameter is set can be specified only once. The exception to this rule is the **WSRR-CUSTOM** parameter, which you can specify a maximum of 255 times.

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 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 (see z/OS Unicode Services User's Guide and Reference). 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 into a CICS TS 3.1 region, you must apply APAR PK23547 to achieve the minimum runtime level of code to install the web service binding file.

CHAR-VARYING={**NO**|**NULL**|**COLLAPSE**|**BINARY**}
: Specifies how character fields in the language structure are mapped when the mapping level is 1.2 or higher. 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. You can select these options:

  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. NO is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping levels 2.0 and earlier.

  This value does not apply to Enterprise and Other PL/I language structures.

  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. NULL is the default value for the **CHAR-VARYING** parameter for C/C++.

  This value does not apply to Enterprise and Other PL/I language structures.

  COLLAPSE
  Character fields are mapped to an <xsd:string>. Trailing white space in the field is not included in the SOAP message. The inbound SOAP message is parsed to remove all leading, trailing, and embedded white

space. COLLAPSE is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping level 2.1 onwards.

For more information about variable-length values and white space, see Support for variable-length values and white space.

BINARY
Character fields are mapped to an <xsd:base64binary> and are processed as fixed-length fields. The BINARY value on the **CHAR-VARYING** parameter is available only at mapping levels 2.1 and onwards.

CHAR-OCCURS={**STRING**|**ARRAY**}
Specifies how character arrays in the language structure are mapped when the mapping level is 4.0 or higher. For example, `PIC X OCCURS 20`. This parameter is only for use by the COBOL language.

ARRAY
Character arrays are mapped to an XML array. This means that every character is mapped as an individual XML element. This is also the behaviour at mapping levels 3.0 and earlier.

STRING
Character arrays are mapped to an XML string. This means that the entire COBOL array is mapped as a single XML element.

CHAR-USAGE={**NATIONAL**|**DBCS**}

In COBOL, the national data type, `PIC N`, can be used for UTF-16 or DBCS data. This setting is controlled by the `NSYMBOL` compiler option. You must set the **CHAR-USAGE** parameter on the assistant to the same value as the `NSYMBOL` compiler option to ensure that the data is handled appropriately. This is typically set to `CHAR-USAGE=NATIONAL` when you use UTF-16.

DBCS Data from `PIC (n)` fields is treated as UTF-16 encoded data.

NATIONAL
Data from `PIC (n)` fields is treated as DBCS encoded data.

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.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**UNUSED**|**PACKED15**}
Specifies if potential ABSTIME fields in the high-level language structure are mapped as timestamps:

PACKED15
Packed decimal fields of length 15 (8 bytes) are treated as CICS ABSTIME fields, and mapped as timestamps.

UNUSED
Packed decimal fields of length 15 (8 bytes) are not treated as timestamps.

You can set this parameter at a mapping level of 3.0.

HTTPPROXY={*domain name*:*port number*|*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 DFHLS2WS uses a proxy server to access the internet, specify the domain name or IP address and the port number of the proxy server. For example:

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

In other cases, this parameter is not required.

HTTPPROXY-PASSWORD=*value*
Specifies the HTTP proxy password that must be used with **HTTPPROXY-USERNAME** if the system on which you are running DFHLS2WS uses a HTTP proxy server to access the Internet, and the HTTP proxy server uses basic authentication. You can use this parameter only when you also specify **HTTPPROXY**.

HTTPPROXY-USERNAME=*value*
Specifies the HTTP proxy username that must be used with **HTTPPROXY-PASSWORD** if the system on which you are running DFHLS2WS uses a HTTP proxy server to access the Internet, and the HTTP proxy server uses basic authentication. You can use this parameter only 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 z/OS UNIX 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.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHLS2WS.

MAPPING-LEVEL={**1.0**|**1.1**|**1.2**|**2.0**|**2.1**|**2.2**|**3.0**|**4.0**}

Specifies the level of mapping that DFHLS2WS uses when generating the web service binding file and web service description. You can select these options:

1.0 This mapping level is the default. It indicates that the web service binding file is generated using CICS TS 3.1 mapping levels.

1.1 Use this mapping to regenerate a binding file at this specific level.

1.2 At this mapping level, you can use the **CHAR-VARYING** parameter to control how character arrays are processed at run time. VARYING and VARYINGZ arrays are also supported in PL/I.

2.0 Use this mapping level in a CICS TS 3.2 region or later to take advantage of the enhancements to the mapping between the language structure and web services binding file.

2.1 Use this mapping level with a CICS TS 3.2 region that has APAR PK59794 applied or with any region later than CICS TS 3.2. At this mapping level you can take advantage of the new values for the **CHAR-VARYING** parameter, COLLAPSE and BINARY. FILLER fields in COBOL and * fields in PL/I are systematically ignored at this mapping level, the fields do not appear in the generated WSDL document, and an appropriate gap is left in the data structures at run time.

2.2 Use this mapping level with a CICS TS 3.2 region that has APAR PK69738 applied or with any region later than CICS TS 3.2 to take advantage of mapping enhancements when using DFHWS2LS.

3.0 Use this mapping level with a CICS TS 4.1 region. At this mapping level you can create a web service from an application that uses many containers in its interface by setting the **REQUEST-CHANNEL** and **RESPONSE-CHANNEL** parameters. You can also map dateTime fields to XML timestamps by setting the **DATETIME** parameter.

4.0 Use this mapping level with a CICS TS 5.2 region. At this mapping level you can use COBOL OCCURS DEPENDING ON fields and the **CHAR-OCCURS** parameter.

For more information about mapping levels, see Mapping levels for the CICS assistants

MINIMUM-RUNTIME-LEVEL={**MINIMUM**|**1.0**|**1.1**|**1.2**|**2.0**|**2.1**|**2.2**|**3.0**|**4.0**|**CURRENT**}

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

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. Some parameters are not available at this runtime level.

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 can use a mapping level of 1.1 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

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 a mapping level of 1.2 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.0 The generated web service binding file deploys successfully into a CICS TS 3.2 region or later. You can use a mapping level of 2.0 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.1 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK59794 applied or into any region later than CICS TS 3.2. You can use a mapping level of 2.1 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

2.2 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK69738 applied or into any region later than CICS TS 3.2. With this runtime level, you can use a mapping level of 2.2 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

3.0 The generated web service binding file deploys successfully into a CICS TS 4.1 region or later. With this runtime level, you can use a mapping level of 3.0 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

OPERATION-NAME=*value*
Specifies the operation name that is used in the generated WSDL document. If no value is supplied, then a default name is generated using the value of the **PGMNAME** parameter followed by value **operation**.

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, the z/OS UNIX System Services code page is used. For example, you might 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.

- At mapping levels earlier than 3.0, the channel can contain only one container, which is used for both input and output. Use the **CONTID** parameter to specify the name of the container. The default name is DFHWS-DATA.
- At mapping level 3.0, the channel can contain multiple containers. Use the **REQUEST-CHANNEL** and **RESPONSE-CHANNEL** parameters. Do not specify **PDSLIB**, **REQMEM**, or **RESPMEM**.

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

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of the CICS PROGRAM resource for the target application program that will be exposed as a web service. The CICS web service support will link to this program.

REQMEM=*value*
Specifies the name of the partitioned data set member that 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.

REQUEST-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when receiving a SOAP message from a web service requester. The channel description is an XML document that must conform to the CICS-supplied channel schema.

You can use this parameter at mapping level 3.0 only.

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 that 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.

Omit this parameter if no response is involved; that is, for one-way messages.

RESPONSE-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when sending a SOAP response message to

a web service requester. The channel description is an XML document that must conform to the CICS-supplied channel schema.

You can use this parameter at mapping level 3.0 only.

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.

SOAPVER={**1.1**|**1.2**|**ALL**}
Specifies the SOAP level to use in the generated web service description. This parameter is available only when the **MINIMUM-RUNTIME-LEVEL** is set to 2.0 or higher.

1.1 The SOAP 1.1 protocol is used as the binding for the web service description.

1.2 The SOAP 1.2 protocol is used as the binding for the web service description.

ALL Both the SOAP 1.1 or 1.2 protocol can be used as the binding for the web service description.

If you do not specify a value for this parameter, the default value depends on the version of WSDL that you want to create:

- If you require only WSDL 1.1, the SOAP 1.1 binding is used.
- If you require only WSDL 2.0, the SOAP 1.2 binding is used.
- If you require both WSDL 1.1 and WSDL 2.0, both SOAP 1.1 and 1.2 bindings are used for each web service description.

SSL-KEYSTORE=*value*
This optional parameter specifies the fully qualified location of the key store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-KEYPWD=*value*
This optional parameter specifies the password for the key store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTSTORE=*value*
This optional parameter specifies the fully qualified location of the trust store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTPWD=*value*
This optional parameter specifies the password for the trust store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

STRUCTURE=(*request*,*response*)

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

request

Specifies the name of the high-level structure that contains 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 sync point.

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

YES The remote web service can issue a sync point. 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- to 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*

This parameter specifies the relative or absolute 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- to 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 z/OS UNIX name of the web service binding file. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is `.wsbind`.

WSDL=*value*
The fully qualified z/OS UNIX name of the file into which the web service description is written. The web service description conforms to the WSDL 1.1 specification. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is `.wsdl`.

WSDL_1.1=*value*
The fully qualified z/OS UNIX name of the file into which the web service description is written. The web service description conforms to the WSDL 1.1 specification. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is `.wsdl`. This parameter produces the same result as the **WSDL** parameter, so you can specify only one or the other.

WSDL_2.0=*value*
The fully qualified z/OS UNIX name of the file into which the web service description is written. The web service description conforms to the WSDL 2.0 specification. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is `.wsdl`. This parameter can be used with the **WSDL** or **WSDL_1.1** parameters. It is available only when the **MINIMUM-RUNTIME-LEVEL** is set to `2.0` or higher.

WSDLCP={**LOCAL**|**UTF-8**|**EBCDIC-CP-US**}
Specifies the code page that is used to generate the WSDL document.

LOCAL
Specifies that the WSDL document is generated using the local code page and no encoding tag is generated in the WSDL document.

UTF-8 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.

EBCDIC-CP-US
This value specifies that the WSDL document is generated using the US EBCDIC code page. An encoding tag is generated in the WSDL document.

WSDL-NAMESPACE=*value*
Specifies the namespace for CICS to use in the generated WSDL document.

If you do not specify this parameter, CICS generates a namespace automatically.

WSRR-CUSTOM-*PropertyName=value*
Use this optional parameter to add customized metadata to the WSDL document in the WSRR. The WSRR-CUSTOM-*PropertyName=value* pairs are added into the WSDL document and appear in WSRR without the WSSR-CUSTOM prefix.

You can specify a maximum of 255 custom *PropertyName=value* pairs. Avoid duplicate and blank *PropertyName=value* pairs.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-DESCRIPTION=*value*
Use this optional parameter to specify the metadata that describes the WSDL document being published.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-ENCODING=*value*
Use this optional parameter to specify the character set encoding of the WSDL document. If the **WSRR-ENCODING** parameter is not specified, WSRR uses the value specified in the WSDL document.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-LOCATION=*value*
Use this optional parameter to specify the URI that identifies the location of the WSDL document. If this parameter is not specified, the URI defaults to the filename specified in the **WSDL** parameter. For example, if the value of the **WSDL** parameter is `wsrr/example.wsdl`, the value of the **WSRR-LOCATION** parameter defaults to `example.wsdl`.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-PASSWORD=*value*
Use this optional parameter if you must enter a password to access WSRR.

If the **WSRR-USERNAME** parameter is specified, you must also specify this parameter.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-SERVER={*domain name*:*port number*|*IP address*:*port number*}
Use this parameter to specify the location of the IBM WebSphere Service Registry and Repository (WSRR) server. If this parameter is specified, WSRR parameter validation is used.

WSRR-USERNAME=*value*
Use this optional parameter if you are required to specify a user name to access WSRR. This user name is used by WSRR to set the owner property.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-VERSION={**1**|*value*}
Use this parameter to set the version property of the WSDL document in WSRR.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

Other information

- The user ID under which DFHLS2SC runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

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/wsbind/example.log
MINIMUM-RUNTIME-LEVEL=2.1
MAPPING-LEVEL=2.1
CHAR-VARYING=COLLAPSE
PGMNAME=DFH0XCMN
URI=http://myserver.example.org:8080/exampleApp/example
PGMINT=COMMAREA
SOAPVER=1.1
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
WSDL=/u/exampleapp/wsdl/example.wsdl
WSDL_2.0=/u/exampleapp/wsdl/example_20.wsdl
WSDLCP=LOCAL
WSDL-NAMESPACE=http://mywsdlnamespace
/*
```

DFHWS2LS: WSDL to high-level language conversion:

The DFHWS2LS procedure 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.

The job control statements for DFHWS2LS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

Job control statements for DFHWS2LS

JOB Starts the job.

EXEC Specifies the procedure name (DFHWS2LS).

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 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/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 z/OS UNIX directory path used on other parameters. The default is the empty string.

Typically, you do not 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 z/OS UNIX that DFHWS2LS uses as a temporary workspace. 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 file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*.

Typically, you do not 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 at run time:

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.

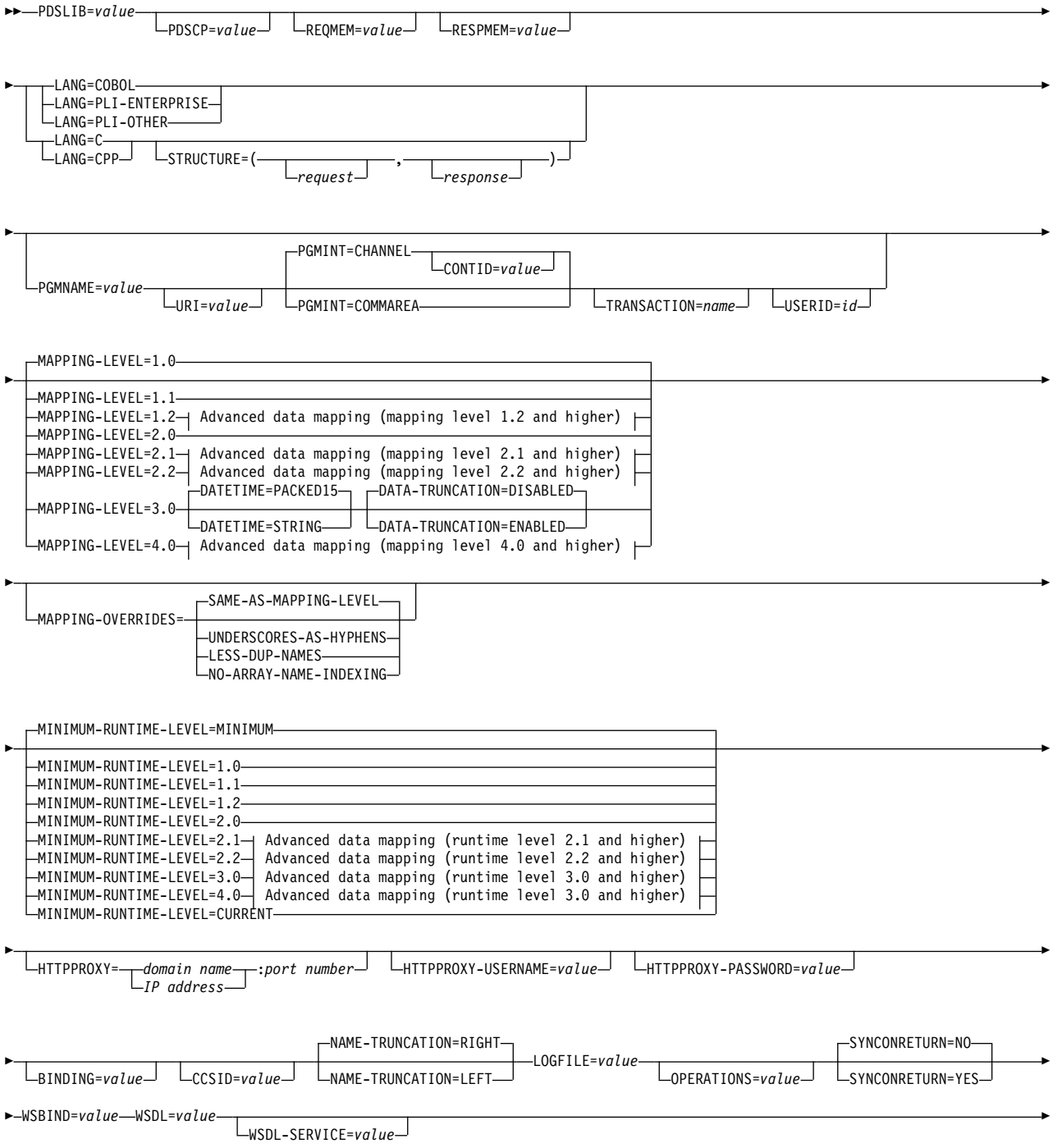
The default names for the files, when **TMPDIR** and **TMPFILE** are not specified, are as follows:

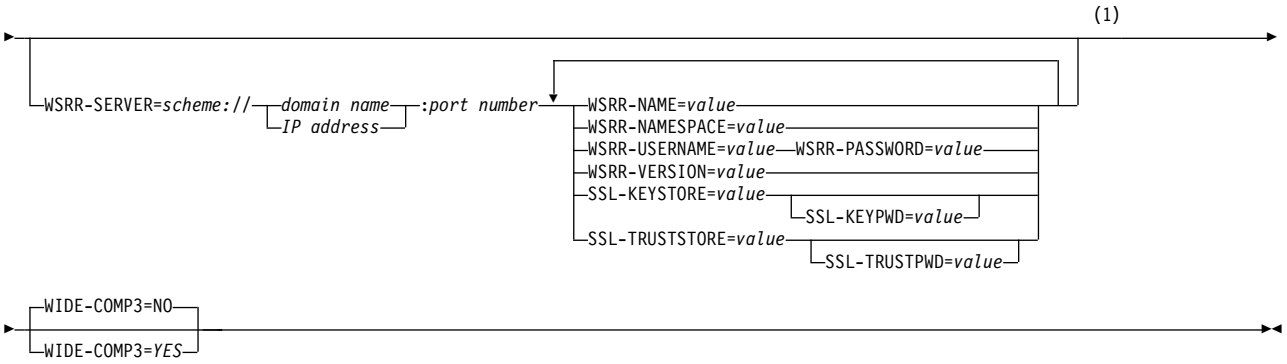
```
/tmp/WS2LS.in
/tmp/WS2LS.out
/tmp/WS2LS.err
```

Important: DFHWS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHWS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job overwriting the workspace files while another job is using them, leading to unpredictable failures.

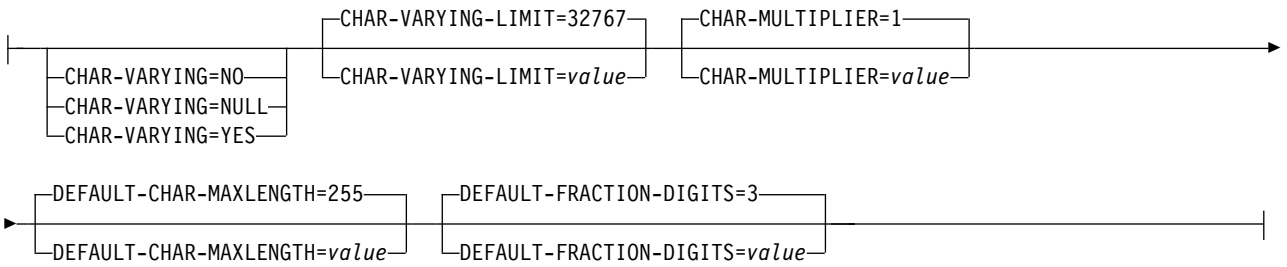
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of the job.

Input parameters for DFHWS2LS



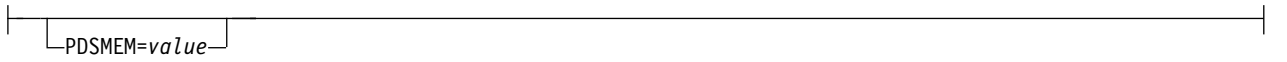


Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):

Advanced data mapping (mapping level 2.2 and higher):



Advanced data mapping (runtime level 2.1 and higher):

Advanced data mapping (runtime level 3.0 and higher):

Notes:

1 Each of the WSRR parameters that can be specified when the **WSRR-SERVER** parameter is set can be specified only once.

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 must 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.
- A comma in the last character position of the line is an optional line separator, and is ignored.

Parameter descriptions

BINDING=*value*
: If the web service description contains more than one <wsdl: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 <wsdl: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 (see z/OS Unicode Services User's Guide and Reference). 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 into a CICS TS 3.1 region, you must apply APAR PK23547 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 or later. The *value* of this parameter can be a positive integer in the range of 1 - 2,147,483,647. 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 might 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.

  When you set **CCSID=1200** (indicating UTF-16), the only valid values for **CHAR-MULTIPLIER** are `2` or `4`. When you use UTF-16, the default value is `2`. Use **CHAR-MULTIPLIER=2** when you expect application data to contain characters that

require 1 UTF-16 encoding unit. Use **CHAR-MULTIPLIER=4** when you expect application data to contain characters that require 2 UTF-16 encoding units.

Note: Setting **CHAR-MULTIPLIER** to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger **CHAR-MULTIPLIER** value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.

CHAR-VARYING={**NO**|**NULL**|**YES**}
Specifies how variable-length character data is mapped when the mapping level is 1.2 or higher. Variable-length binary data types are always mapped to either a container or a varying structure. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

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 two related elements: data-length and the data.

CHAR-VARYING-LIMIT={**32767**|*value*}
Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure when the mapping level is 1.2 or higher. If the character or binary 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 32,767 bytes.

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.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**PACKED15**|**STRING**}
Specifies how <xsd:dateTime> elements are mapped to the language structure.

PACKED15
The default is that any <xsd:dateTime> element is processed as a timestamp and is mapped to CICS ABSTIME format.

STRING
The <xsd:dateTime> element is processed as text.

DEFAULT-CHAR-MAXLENGTH={**255**|*value*}
Specifies the default array length of character data in characters for mappings where no length is implied in the web service description document, when the mapping level is 1.2 or higher. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647.

DEFAULT-FRACTION-DIGITS={**3**|*value*}
Specifies the default number of fraction digits to use on an XML decimal schema type. The default is 3. For COBOL, the valid range is 0-17, or 0-30 if parameter **WIDE-COMP3** is being used. For C or PLI the valid range is 0-30.

HTTPPROXY={*domain name*:*port number*|*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 the port number of the proxy server. For example:

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

In other cases, this parameter is not required.

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

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

INLINE-MAXOCCURS-LIMIT={**1**|*value*}
Specifies whether or not inline variable repeating content is used based on the `maxOccurs` attribute. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.

The **INLINE-MAXOCCURS-LIMIT** parameter is available only at mapping level 2.1 onwards. The value of **INLINE-MAXOCCURS-LIMIT** can be a positive integer in the range of 0 - 32,767. A value of `0` indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the *value* of the `maxOccurs` attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise inline mapping is used.

When deciding if you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.

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 z/OS UNIX 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.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHWS2LS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHWS2LS uses when generating the web service binding file and language structure. You can select these options:

1.0 The web service binding file and language structure are generated using CICS TS 3.1 mapping levels.

1.1 XML attributes and <list> and <union> data types are mapped to the language structure. Character and binary data that have a maximum length of more than 32,767 bytes are mapped to a container. The container name is created in the language structure.

1.2 Use the **CHAR-VARYING** and **CHAR-VARYING-LIMIT** parameters 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 have a maximum length of less than 32,768 bytes are mapped to a VARYING structure for all languages except C++, where character data is mapped to a null-terminated string.

2.0 Use this mapping level in a CICS TS 3.2 region or later to take advantage of the enhancements to the mapping between the language structure and web services binding file.

2.1 Use this mapping level with a CICS TS 3.2 region that has APAR PK59794 applied, or any region later than CICS TS 3.2 for <xsd:any> and xsd:anyType support, the option to map variably repeating content inline with the **INLINE-MAXOCCURS-LIMIT** parameter, and support for minOccurs="0" on <xsd:sequence>, <xsd:choice>, and <xsd:all>.

2.2 Use this mapping level with a CICS TS 3.2 region that has APAR PK69738 applied or with any region later than CICS TS 3.2. It provides the following support:

- Elements with fixed values
- Enhanced support for <xsd:choice> elements

- Abstract data types
- Abstract elements
- Substitution groups

3.0 Use this mapping level with a CICS TS 4.1 or later region. At this mapping level you can transform timestamps to CICS ABSTIME format.

4.0 Use this mapping level with a CICS TS 5.2 region when you want to use UTF-16.

For more information about mapping levels, see Mapping levels for the CICS assistants.

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|LESS-DUP-NAMES|NO-ARRAY-NAME-INDEXING}

Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

Note: Any of the sub options may be used in a comma delimited list. The options are not mutually exclusive, they are combinatorial and unordered.

LESS-DUP-NAMES

This parameter generates non-structural structure field names with `_value` at the end of the name to enable direct referencing to the field. For example, in the following PLI language structure, when MAPPING-OVERRIDES=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with `_value`:

```
09 streetName,
    12 streetName                    CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

The resulting structure is as follows:

```
09 streetName,
    12 streetName_value              CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

SAME-AS-MAPPING-LEVEL

This parameter generates language structures in the same style as the mapping level. This is the default.

UNDERSCORES-AS-HYPHENS

For COBOL only. This parameter converts any underscores in the WSDL document to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure they are unique. For more information, see XML schema to COBOL mapping in Developing applications.

NO-ARRAY-NAME-INDEXING

For Enterprise PL/I only. Ensures that the field names within an array are unique only within the scope of the higher level structure.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}

Specifies the minimum CICS runtime environment into which the web service

binding file can be deployed. If you select a level that does not match the other parameters that you have specified, you receive an error message. You can select these options:

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. Some parameters are not available at this runtime level.

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 can use a mapping level of 1.1 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

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 a mapping level of 1.2 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.0 The generated web service binding file deploys successfully into a CICS TS 3.2 region or later. You can use a mapping level of 2.0 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.1 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK59794 applied or into any region later than CICS TS 3.2. You can use a mapping level of 2.1 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

2.2 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK69738 applied or into any region later than CICS TS 3.2. With this runtime level, you can use a mapping level of 2.2 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

3.0 The generated web service binding file deploys successfully into a CICS TS 4.1 region or later. With this runtime level, you can use a mapping level of 3.0 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

NAME-TRUNCATION={LEFT|RIGHT}
Specifies whether XML element names are truncated from the left or the right. The CICS web services assistant truncates XML element names to the appropriate length for the high-level language specified; by default names are truncated from the right.

OPERATIONS=*value*
For web service requester applications, specifies a subset of valid

<wsdl:Operation> elements from the web service description that is used to generate the web service binding file. Each <wsdl:Operation> element is separated by a space; the list can span more than one line if necessary. You can use this parameter for both WSDL 1.1 and WSDL 2.0 documents.

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, the z/OS UNIX System Services code page is used. For example, you might specify **PDSCP**=037.

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.

PDSMEM=*value*
Specifies a 1- to 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 abstract data types. It generates the member name by appending a 2-digit number to the prefix.

Use this parameter at a mapping level of 2.2 or higher for naming the language structures associated with abstract data types. If the **PDSMEM** parameter is omitted, language structures for abstract data types are named using the value in the **REQMEM** parameter.

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.

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of a CICS PROGRAM resource.

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

When DFHWS2LS is used to generate a web service binding file that will be used in a service requester, omit this parameter.

REQMEM=*value*

Specifies a 1- to 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 2-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- to 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 2-digit number to the prefix.

Omit this parameter if no response is involved; that is, for one-way messages.

SSL-KEYSTORE=*value*

This optional parameter specifies the fully qualified location of the key store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-KEYPWD=*value*

This optional parameter specifies the password for the key store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTSTORE=*value*

This optional parameter specifies the fully qualified location of the trust store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTPWD=*value*

This optional parameter specifies the password for the trust store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

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 sync point.

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

YES The remote web service can issue a sync point. 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- to 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 uses 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. CICS does not generate a URIMAP resource for a service requester. You can define your own URIMAP resource for service requesters to use when they make client requests to the URI of the target web service. When a service requester issues the **INVOKE SERVICE** command, CICS uses the soap:address location from the wsdl:port specified in the web service description if present. You can override that and specify a different URI using the URIMAP or URI options on the **INVOKE SERVICE** command.

USERID=*id*
In a service provider, this parameter specifies a 1- to 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 $ @ #

WIDE-COMP3={**NO**|**YES**}
For COBOL only. Controls the maximum size of the packed decimal variable length in the COBOL language structure.

NO DFHWS2LS limits the packed decimal variable length to 18 when generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.

YES DFHWS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

WSADDR-EPR-ANY={**TRUE**|**FALSE**}
Specifies whether CICS transforms a WS-Addressing endpoint reference (EPR) into its components parts in the language structures or treats the EPR as an <xsd:any> type. Treating the EPR as an <xsd:any> type means that the **WSACONTEXT BUILD** API can use the EPR XML directly.

FALSE
DFHWS2LS behaves typically, transforming the XML to a high-level language structure.

TRUE Setting this option to `TRUE` means that at run time CICS treats the whole EPR as an <xsd:any> type and places the EPR XML into a container that can be referenced by the application. The application can use the EPR XML with the **WSACONTEXT BUILD** API to construct an EPR in the addressing context.

This parameter is available only at runtime level 3.0 onwards.

WSBIND=*value*
The fully qualified z/OS UNIX name of the web service binding file. DFHWS2LS creates the file, but not the directory structure, if it does not already exist. The file extension defaults to `.wsbind`.

WSDL=*value*
The fully qualified z/OS UNIX name of the file that contains the web service description. If you are using WSRR to retrieve the WSDL document, this parameter specifies the location on the file system to which a local copy of the WSDL document will be written.

WSDL-SERVICE=*value*
Specifies the `wsdl:Service` element that is used when the web service description contains more than one Service element for a Binding element. If you specify a value for the **BINDING** parameter, the Service element that you specify for this parameter must be consistent with the specified Binding element. You can use this parameter with either WSDL 1.1 or WSDL 2.0 documents.

WSRR-NAME=*value*
Specifies the name of the WSDL document to retrieve from WSRR. Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-NAMESPACE=*value*
Specifies the namespace of the WSDL document to retrieve from WSRR. You can optionally use this parameter when the **WSRR-SERVER** parameter is specified to fully qualify the WSDL document name specified in the **WSRR-NAME** parameter.

WSRR-PASSWORD=*value*
Use this optional parameter if you must enter a password to access WSRR.

If the **WSRR-USERNAME** parameter is specified, you must also specify this parameter.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-SERVER={*domain name*:*port number*|*IP address*:*port number*}
Use this parameter to specify the location of the IBM WebSphere Service Registry and Repository (WSRR) server. If this parameter is specified, WSRR parameter validation is used.

WSRR-USERNAME=*value*
Use this optional parameter if you are required to specify a user name to access WSRR. This user name is used by WSRR to set the owner property.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-VERSION=*value*
Specifies the version of the WSDL document to retrieve from WSRR. You can use this parameter only when the **WSRR-SERVER** parameter is specified.

XML-ONLY={**TRUE**|**FALSE**}
Specifies whether or not CICS transforms the XML in the SOAP message to application data. Use the **XML-ONLY** parameter to write web service applications that process the XML themselves.

TRUE CICS does not perform any transformations to the XML. The service requester or provider application must work with the contents of the DFHWS-BODY container directly to map data between XML and the high-level language.

FALSE
CICS transforms the XML to a high-level language.

This parameter is available only at runtime level 2.1 onwards.

Other information

- The user ID under which DFHLS2SC runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement. For more information, see z/OS UNIX System Services User's Guide.

Example

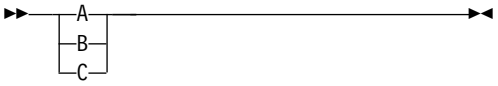
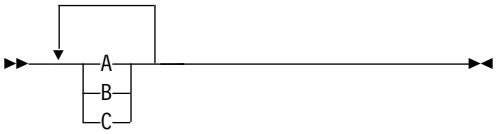
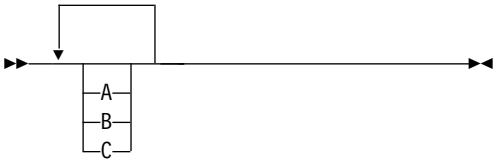
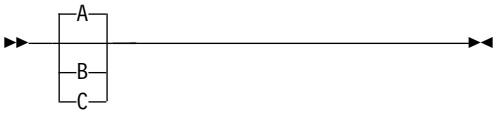
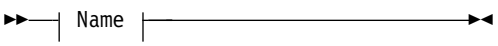
```
//WS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHWS2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=3.0
MAPPING-OVERRIDES=UNDERSCORES-AS-HYPHENS
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example
```

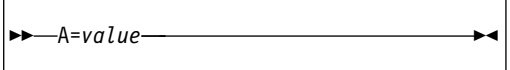
```
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
WSDL=/u/exampleapp/wsdl/example.wsdl
/*
```

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.
	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.
 **Name:** 	A reference to a named section of syntax notation.

Notation	Explanation
	**A**= denote characters that should be entered exactly as shown. *value* denotes a variable, for which you should specify an appropriate value.

Mapping levels for the CICS assistants:

A mapping is the set of rules that specifies how information is converted between language structures and XML schemas. To benefit from the most sophisticated mappings available, you are recommended to set the **MAPPING-LEVEL** parameter in the CICS assistants to the latest level.

Each level of mapping inherits the function of the previous mapping, with the highest level of mapping offering the best capabilities available. The highest mapping level provides more control over data conversion at run time and removes restrictions on support for certain data types and XML elements.

You can set the **MAPPING-LEVEL** parameter to an earlier level if you want to redeploy applications that were previously enabled at that level.

Mapping level 3.0

Mapping level 3.0 is compatible with a CICS TS 4.1 region.

This mapping level provides the following support:

- DFHSC2LS and DFHWS2LS map `xsd:dateTime` data types to CICS ASKTIME format.
- DFHLS2WS can generate a WSDL document and web service binding from an application that uses many containers rather than just one container.
- Tolerating truncated data that is described by a fixed length data structure. You can set this behavior by using the **DATA-TRUNCATION** parameter on the CICS assistants.

Mapping level 2.2 and higher

Mapping level 2.2 is compatible with a CICS TS 3.2 region, with APAR PK69738 applied, and higher.

At mapping level 2.2 and higher, DFHSC2LS and DFHWS2LS support the following XML mappings:

- Fixed values for elements
- Substitution groups
- Abstract data types
- XML schema <sequence> elements can nest inside <choice> elements

DFHSC2LS and DFHWS2LS provide enhanced support for the following XML mappings:

- Abstract elements
- XML schema <choice> elements

Mapping level 2.1 and higher

Mapping level 2.1 is compatible with a CICS TS 3.2 region, with APAR PK59794 applied, and higher.

This mapping level includes greater control over the way variable content is handled with the new **INLINE-MAXOCCURS-LIMIT** parameter and new values on the **CHAR-VARYING** parameter.

At mapping level 2.1 and higher, DFHSC2LS and DFHWS2LS offer the following new and improved support for XML mappings:

- The XML schema <any> element
- The `xsd:anyType` type
- Toleration of abstract elements
- The **INLINE-MAXOCCURS-LIMIT** parameter
- The `minOccurs` attribute

The **INLINE-MAXOCCURS-LIMIT** parameter specifies whether variably repeating lists are mapped inline. For more information on mapping variably repeating content inline, see Variable arrays of elements.

Support for the `minOccurs` attribute has been enhanced on the XML schema <sequence>, <choice>, and <all> elements. If `minOccurs="0"`, the CICS assistant treats these element as though the `minOccurs="0"` attribute is also an attribute of all its child elements.

At mapping level 2.1 and higher, DFHLS2SC and DFHLS2WS support the following XML mappings:

- FILLER fields in COBOL and PL/I are ignored
- A value of COLLAPSE for the **CHAR-VARYING** parameter
- A value of BINARY for the **CHAR-VARYING** parameter

FILLER fields in COBOL and PL/I are ignored; they do not appear in the generated XML schema and an appropriate gap is left in the data structures at run time.

`COLLAPSE` causes CICS to ignore trailing spaces in text fields.

`BINARY` provides support for binary fields. This value is useful when converting COBOL into an XML schema. This option is available only on SBCS character arrays and allows the array to be mapped to fixed-length `xsd:base64Binary` fields rather than to `xsd:string` fields.

Mapping level 1.2 and higher

Mapping level 1.2 is compatible with a CICS TS 3.1 region and higher.

Greater control is available over the way character and binary data are transformed at run time with these additional parameters on the batch tools:

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**
- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

If you decide to use the **CHAR-MULTIPLIER** parameter in DFHSC2LS or DFHWS2LS, note that the following rules apply after the value of this parameter is used to calculate the amount of space required for character data.

- DFHSC2LS and DFHWS2LS provide these mappings:
  - Variable-length character data types that have a maximum length of more than 32 767 bytes map 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.
  - Variable-length character data types that have a maximum length of less than 32 768 bytes map 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 the way that variable-length character data is mapped.
  - Variable-length binary data that has a maximum length of less than 32 768 bytes maps 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.

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 or DFHSC2LS.

DFHLS2SC and DFHLS2WS provide these mappings:

- 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 the way that variable-length character data is handled at run time for all languages except PL/I.
- Base64Binary data types map 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.

Mapping level 1.1 and higher

Mapping level 1.1 is compatible with a CICS TS 3.1 region and higher.

This mapping level provides improved mapping of XML character and binary data types, in particular when mapping data of variable length that has `maxLength` and `minLength` attributes defined with different values in the XML schema. Data is handled in the following ways:

- Character and binary data types that have a fixed length that is greater than 16 MB 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.

  Because 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 XML schema or web service description. If the length of

the data is significant, you can either write your application to check it or turn validation on in the CICS region. Both SOAP and XML validation have a significant performance impact.

- XML schema <list> and <union> data types map to character fields.
- Schema-defined XML attributes are mapped rather than ignored. A maximum of 255 attributes is allowed for each XML element. See Support for XML attributes for further information.
- The `xsi:nil` attribute is supported. See Support for XML attributes for further information.

Mapping level 1.1 only

Mapping level 1.1 is compatible with a CICS TS 3.1 region and higher.

This mapping level provides improved mapping of XML character and binary data types, in particular when mapping data of variable length that has `maxLength` and `minLength` attributes 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 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 of less than 16 MB 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.
- CICS encodes and decodes data in the hexBinary format but not in base64Binary format. 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.
  - ceil(x) is the smallest integer greater than or equal to x.

  If the length of z is greater than 24 566 bytes, the resulting language structure fails to compile. If you have base64Binary data that is greater than 24 566 bytes, you are recommended to use a mapping level of 1.2. With mapping level 1.2, you can map the base64Binary data to a container instead of using a field in the language structure.

Mapping level 1.0 only

Mapping level 1.0 is compatible with a CICS TS 3.1 region and higher.

Note the following limitations, which have been modified in later mapping levels:

- DFHSC2LS and DFHWS2LS map character and binary data types in the XML schema to fixed-length fields in the language structure. Look at this partial XML schema:

```
<xsd:element name="example">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
```

```
      <xsd:maxLength value="33000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

That partial XML schema appears in a COBOL language structure like this:

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

- CICS encodes and decodes data in the hexBinary format but not in base64Binary format. DFHSC2LS and DFHWS2LS map Base64Binary data to a fixed-length character field, the contents of which must be encoded or decoded by the application program.
- DFHSC2LS and DFHWS2LS ignore XML attributes during processing.
- DFHLS2SC and DFHLS2WS interpret character and binary fields in the language structure as fixed-length fields and map those 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 insufficient data is available.

High-level language and XML schema mapping:

Use the CICS assistants to generate mappings between high-level language structures and XML schemas or WSDL documents. The CICS assistants also generate XML schemas or WSDL documents from high-level language data structures, or vice-versa.

Utility programs DFHSC2LS and DFHLS2SC are collectively known as the CICS XML assistant. Utility programs DFHWS2LS and DFHLS2WS are collectively known as the CICS Web services assistant.

- DFHLS2SC and DFHLS2WS map high-level language structures to XML schemas and WSDL documents respectively.
- DFHSC2LS and DFHWS2LS map XML schemas and WSDL documents to high-level language structures.

The two mappings are not symmetrical:

- If you process a language data structure with DFHLS2SC or DFHLS2WS and then process the resulting XML schema or WSDL document with the complementary utility program (DFHSC2LS or DFHWS2LS respectively), do not expect the final data structure to be the same as the original. However, the final data structure is logically equivalent to the original.
- If you process an XML schema or WSDL document with DFHSC2LS or DFHWS2LS and then process the resulting language structure with the complementary utility program (DFHLS2SC or DFHLS2WS respectively), do not expect the XML schema in the final XML schema or WSDL document to be the same as the original.
- In some cases, DFHSC2LS and DFHWS2LS generate language structures that are not supported by DFHLS2SC and DFHLS2WS.

You must code language structures processed by DFHLS2SC and DFHLS2WS according to the rules of the language, as implemented in the language compilers that CICS supports.

Data mapping limitations when using the CICS assistants:

CICS supports bidirectional data mappings between high-level language structures and XML schemas or WSDL documents that conform to WSDL version 1.1 or 2.0,

with certain limitations. These limitations apply only to the DFHWS2LS and DFHSC2LS tools and vary according to the mapping level.

Limitations at all mapping levels

- Only SOAP bindings that use literal encoding are supported. Therefore, you must set the `use` attribute to a value of `literal`; `use="encoded"` is not supported.
- Data type definitions must be encoded using the XML Schema Definition language (XSD). In the schema, data types used in the SOAP message must be explicitly declared.
- The length of some keywords in the web services description is limited. For example, operation, binding, and part names are limited to 255 characters. In some cases, the maximum operation name length might be slightly shorter.
- 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.
- DFHWS2LS and DFHSC2LS support only a single <xsd:any> element in a particular scope. For example, the following schema fragment is not supported:

  ```
  <xsd:sequence>
   <xsd:any/>
   <xsd:any/>
  </xsd:sequence>
  ```

  Here, `<xsd:any>` can specify `minOccurs` and `maxOccurs` if required. For example, the following schema fragment is supported:

  ```
  <xsd:sequence>
   <xsd:any minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
  ```

- Cyclic references are not supported. For example, where type A contains type B which, in turn, contains type A.
- Recurrence is not supported in group elements, such as <xsd:choice>, <xsd:sequence>, <xsd:group>, or <xsd:all> elements. For example, the following schema fragment is not supported:

  ```
  <xsd:choice maxOccurs="2">
     <xsd:element name="name1" type="string"/>
  </xsd:choice>
  ```

  The exception is at mapping level 2.1 and higher, where `maxOccurs="1"` and `minOccurs="0"` are supported on these elements.
- DFHSC2LS and DFHWS2LS do not support data types and elements in the SOAP message that are derived from the declared data types and elements in the XML schema either from the `xsi:type` attribute or from a substitution group, except at mapping level 2.2 and higher if the parent element or type is defined as abstract.
- Embedded <xsd:sequence> and <xsd:group> elements inside an <xsd:choice> element are not supported prior to mapping level 2.2. Embedded <xsd:choice> and <xsd:all> elements inside an <xsd:choice> element are never supported.

Improved support at mapping level 1.1 and higher

When the mapping level is 1.1 or higher, DFHWS2LS provides support for the following XML elements and element type:

- The <xsd:list> element.
- The <xsd:union> element.
- The `xsd:anySimpleType` type.

- The <xsd:attribute> element. At mapping level 1.0 this element is ignored.

Improved support at mapping level 2.1 and higher

When the mapping level is 2.1 or higher, DFHWS2LS supports the following XML elements and element attributes:

- The <xsd:any> element.
- The `xsd:anyType` type.
- Abstract elements. In earlier mapping levels, abstract elements are supported only as nonterminal types in an inheritance hierarchy.
- The `maxOccurs` and `minOccurs` attributes on the <xsd:all>, <xsd:choice>, and <xsd:sequence> elements, only when `maxOccurs="1"` and `minOccurs="0"`.
- "FILLER" fields in COBOL and "*" fields in PL/I are suppressed. The fields do not appear in the generated WSDL and an appropriate gap is left in the data structures at run time.

Improved support at mapping level 2.2 and higher

When the mapping level is 2.2 or higher, DFHSC2LS and DFHWS2LS provide improved support for the <xsd:choice> element, supporting a maximum of 255 options in the <xsd:choice> element. For more information on <xsd:choice> support, see "Support for `<xsd:choice>`" on page 120.

At mapping level 2.2 and higher, the CICS assistants support the following XML mappings:

- Substitution groups
- Fixed values for elements
- Abstract data types

Embedded <xsd:sequence> and <xsd:group> elements inside an <xsd:choice> element are supported at mapping level 2.2 and higher. For example, the following schema fragment is supported:

```
<xsd:choice>
   <xsd:element name="name1" type="string"/>
   <xsd:sequence/>
</xsd:choice>
```

If the parent element or type in the SOAP message is defined as abstract, DFHSC2LS and DFHWS2LS support data types and elements that are derived from the declared data types and elements in the XML schema.

Improved support at mapping level 3.0 and higher

When the mapping level is 3.0 or higher, the CICS assistants support the following mapping improvements:

- DFHSC2LS and DFHWS2LS map `xsd:dateTime` data types to CICS ASKTIME format.
- DFHLS2WS can generate a WSDL document and web service binding from an application that uses many containers rather than just one container.
- Tolerating truncated data that is described by a fixed length data structure. You can set this behavior by using the **`DATA-TRUNCATION`** parameter on the CICS assistants.

COBOL to XML schema mapping:

The DFHLS2SC and DFHLS2WS utility programs support mappings between COBOL data structures and XML schema definitions.

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 uppercase characters are converted to lowercase.

   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`.

CICS 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 DFHLS2SC or 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 supported only when the mapping level specified in DFHLS2SC or DFHLS2WS is 1.2 or higher.
- SEPARATE CHARACTER is supported at a mapping level of 1.2 or higher 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 or higher:

  COMPUTATIONAL-1

  COMPUTATIONAL-2

- The only PICTURE characters supported for DISPLAY and COMPUTATIONAL-5 data description items are 9, S, and Z.
- The PICTURE characters supported for PACKED-DECIMAL data description items are 9, S, V, and Z.
- The only PICTURE characters supported for edited numeric data description items are 9 and Z.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an `xsd:string` and are processed as null-terminated strings.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to BINARY, character arrays are mapped to `xsd:base64Binary` and are processed as binary data.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to COLLAPSE, trailing white space is ignored for strings.

COBOL data description	Schema simpleType

`PIC X(n)`
`PIC A(n)`
`PIC G(n) DISPLAY-1`
`PIC N(n)`
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:maxlength value="n"/>`
	`  <xsd:whiteSpace value="preserve"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	where *m* =*n*

`PIC S9 DISPLAY`
`PIC S99 DISPLAY`
`PIC S999 DISPLAY`
`PIC S9999 DISPLAY`
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:short">`
	`  <xsd:minInclusive value="-n"/>`
	`  <xsd:maxInclusive value="n"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	where *n* is the maximum value that can be represented by the pattern of '9' characters.

`PIC S9(z) DISPLAY`
 	
where 5 ≤ *z* ≤ 9
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:int">`
	`  <xsd:minInclusive value="-n"/>`
	`  <xsd:maxInclusive value="n"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	where *n* is the maximum value that can be represented by the pattern of '9' characters.

`PIC S9(z) DISPLAY`
 	
where 9 < *z*
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:long">`
	`  <xsd:minInclusive value="-n"/>`
	`  <xsd:maxInclusive value="n"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	where *n* is the maximum value that can be represented by the pattern of '9' characters.

COBOL data description	Schema simpleType
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC 9(*z*) DISPLAY where 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC 9(*z*) DISPLAY where 9 < *z*	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY where *n* ≤ 4.	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY where 5 ≤ *n* ≤ 9.	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY where 9 <*n*.	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY where *n* ≤ 4.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>

COBOL data description	Schema simpleType
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY where 5 ≤ *n* ≤ 9.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY where 9 <*n*.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>
PIC S9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> where *p* = *m* + *n*.
PIC 9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> <xsd:minInclusive value="0"/> </xsd:restriction> </xsd:simpleType> where *p* = *m* + *n*.
PIC S9(*m*) COMP-3 Supported at mapping level 3.0 when DATETIME=PACKED15	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> The format of the timestamp is CICS ABSTIME.
PIC S9(*m*)V9(*n*) DISPLAY Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> where *p* = *m* + *n*.
COMP-1 Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
COMP-2 Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

XML schema to COBOL mapping:

The DFHSC2LS and DFHWS2LS utility programs support mappings between XML schema definitions and COBOL data structures.

The CICS assistants generate 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. You can use the **MAPPING-OVERRIDES** parameter to change the way other characters are handled. For example, if you set the value UNDERSCORES-AS-HYPHENS, any underscore in the XML is converted to a hyphen instead of an X. So monthly_total becomes monthly-total.

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 on an xsd:element 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 with different values.

   For more information, see "Variable arrays of elements" on page 112.

7. For attributes, the previous rules are applied to the element name. The prefix attr- is added to the element name, and is followed by -value or -exist. If the total length is longer than 28 characters, the element name is truncated. For more information, see "Support for XML attributes" on page 116.

   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 30 characters or less.

DFHSC2LS and DFHWS2LS map schema types to COBOL data description elements by using the specified mapping level according to the following table. 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 and an extra character is allocated for the null-terminator.
- 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

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anyType">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 2.0 and below:
	Not supported
	
	Mapping level 2.1:
	
	Supported

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anySimpletype">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 1.0:
	Not supported
	
	Mapping level 1.1 and higher:
	
	`PIC X(255)`

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:`*type*`"`
`  <xsd:length value="`*z*`"/>`
`  </xsd:restriction>`
`</xsd:simpleType>`
 	
where *type* is one of:
`string`
`normalizedString`
`token`
`Name`
`NMTOKEN`
`language`
`NCName`
`ID`
`IDREF`
`ENTITY`
`hexBinary`
	All mapping levels:
	`PIC X(`*z*`)`

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:`*type*`"`
`  </xsd:restriction>`
`</xsd:simpleType>`
 	
where *type* is one of:
`duration`
`date`
`time`
`gDay`
`gMonth`
`gYear`
`gMonthDay`
`gYearMonth`
	All mapping levels:
	`PIC X(32)`

Schema simple type	COBOL data description
<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType>	Mapping level 1.2 and below: PIC X(32) Mapping level 2.0 and higher: PIC X(40) Mapping level 3.0 and higher: PIC S9(15) COMP-3 The format is CICS ABSTIME.

<xsd:simpleType>
  <xsd:restriction base="xsd:*type*">
  </xsd:restriction>
</xsd:simpleType>
 	
where *type* is one of:
  byte
  unsignedByte
	All mapping levels:
	PIC X DISPLAY

<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9999 COMP-5 SYNC or PIC S9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9999 COMP-5 SYNC or PIC 9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9(18) COMP-3
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9(9) COMP-5 SYNC or PIC S9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9(9) COMP-5 SYNC or PIC 9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9(18) COMP-5 SYNC or PIC S9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9(18) COMP-5 SYNC or PIC 9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*m*" <xsd:fractionDigits value="*n*" </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9(*p*)V9(*n*) COMP-3 where $p = m - n$.

Schema simple type	COBOL data description
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC X DISPLAY The value x'00' implies false, x'01' implies true.
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: PIC X(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: PIC X(255)
<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> where the length is not defined.	Mapping level 1.0: Not supported Mapping level 1.1: PIC X(*y*) where $y = 4 \times (\text{ceil}(z/3))$. ceil($x$) is the smallest integer greater than or equal to x. Mapping level 1.2 and higher: PIC X(*z*) where the length is fixed. PIC X(16) where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.1 and below: PIC X(32) Mapping level 1.2 and higher: COMP-1
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.1 and below: PIC X(32) Mapping level 1.2 and higher: COMP-2

Some of the schema types shown in the table map to a COBOL format of COMP-5 SYNC or of DISPLAY, depending on the values (if any) that 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 '9's.

- 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 '9's.

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

C and C++ to XML schema mapping:

The DFHLS2SC and DFHLS2WS utility programs support mappings between C and C++ data types and XML schema definitions.

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`.

DFHLS2SC and DFHLS2WS map 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 DFHLS2SC or DFHLS2WS. The `_Packed` qualifier is supported for structures. These restrictions 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 characters are ignored if they are present in the header file.
  Storage class specifiers:
  `auto`
  `register`
  `static`
  `extern`
  `mutable`
  Qualifiers
  `const`
  `volatile`
  `_Export` (C++ only)
  Function specifiers
  `inline` (C++ only)
  `virtual` (C++ only)
  Initial values
- The header file must not contain these items:
  Unions
  Class declarations

Enumeration data types

Pointer type variables

Template declarations

Predefined macros; that is, macros with names that 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 must use a 32-bit compiler to ensure that an `int` maps to 4 bytes.
- The following C++ reserved keywords are not supported:

  `explicit`

  `using`

  `namespace`

  `typename`

  `typeid`
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an `xsd:string` and are processed as null-terminated strings.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to BINARY, character arrays are mapped to `xsd:base64Binary` and are processed as binary data.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to COLLAPSE, `<xsd:whiteSpace value="collapse"/>` is generated for strings.

C and C++ data type	Schema simpleType

`char[z]`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:length value="z"/>`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`char[8]`
 	
Supported at mapping level 3.0 and higher when DATETIME=PACKED15
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:dateTime"`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	The format of the time stamp is CICS ABSTIME.

`char`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:byte">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`unsigned char`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:unsignedByte">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

C and C++ data type	Schema simpleType
short	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>
unsigned short	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>
int long	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>
unsigned int unsigned long	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>
long long	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>
unsigned long long	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>
bool (C++ only)	<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpletype>
float Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
double Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

XML schema to C and C++ mapping:

The DFHSC2LS and DFHWS2LS utility programs support mappings between the XML schema definitions that are included in each web service description and C and C++ data types.

The CICS assistants generate 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, it is replaced by a leading 'X'.

   For example, `_monthlysummary` becomes `Xmonthlysummary`.
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 one or more 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 112.

6. For attributes, the previous rules are applied to the element name. The prefix `attr_` is added to the element name, and it 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.

DFHSC2LS and DFHWS2LS map schema types 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 and an extra character is allocated for the null-terminator.
- 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	C and C++ data type

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anyType">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 2.0 and below:
	Not supported
	
	Mapping level 2.1 and higher:
	
	Supported

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anySimpletype">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 1.0:
	Not supported
	
	Mapping level 1.1 and higher:
	
	`char[255]`

Schema simpleType	C and C++ data type
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <xsd:length value="*z*"/> </xsd:restriction> </xsd:simpleType> where *type* is one of: string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY hexBinary	All mapping levels: char[*z*]
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> where *type* is one of: duration date decimal time gDay gMonth gYear gMonthDay gYearMonth	All mapping levels: char[32]
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.2 and below: char[32] Mapping level 2.0 and higher: char[40] Mapping level 3.0 and higher: char[8] The format of the time stamp is CICS ABSTIME.
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpletype>	All mapping levels: signed char
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpletype>	All mapping levels: char

Schema simpleType	C and C++ data type
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>	All mapping levels: short
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>	All mapping levels: unsigned short
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpletype>	All mapping levels: char[33]
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>	All mapping levels: int
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>	All mapping levels: unsigned int
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>	All mapping levels: long long
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>	All mapping levels: unsigned long long
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpletype>	All mapping levels: bool (C++ only) short (C only)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: char[255]
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: char[255]

Schema simpleType	C and C++ data type
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: char[y] where y =4×(ceil(z/3)). ceil(x) is the smallest integer greater than or equal to x.
<xsd:simpleType> <xsd:restriction base="xsd:base64binary"> </xsd:restriction> </xsd:simpletype> where the length is not defined	Mapping level 1.2 and higher: 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.
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.1 and below: char[32] Mapping level 1.2 and higher: float(*)
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.0 and below: char[32] Mapping level 1.2 and higher: double(*)

PL/I to XML schema mapping:

The DFHLS2SC and DFHLS2WS utility programs support mappings between PL/I data structures and XML schema definitions. Because the Enterprise PL/I compiler and older PL/I compilers differ, two language options are supported: `PLI-ENTERPRISE` and `PLI-OTHER`.

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`.

DFHLS2SC and DFHLS2WS map 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 DFHLS2SC or DFHLS2WS. The following restrictions also apply:

- Data items with the COMPLEX attribute are not supported.
- Data items with the FLOAT attribute are supported at a mapping level of 1.2 or higher. Enterprise PL/I FLOAT IEEE is not supported.
- VARYING and VARYINGZ pure DBCS strings are supported at a mapping level of 1.2 or higher.
- 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.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an `xsd:string` and are processed as null-terminated strings; this mapping does not apply for Enterprise PL/I.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to BINARY, character arrays are mapped to `xsd:base64Binary` and are processed as binary data.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to COLLAPSE, `<xsd:whiteSpace value="collapse"/>` is generated for strings.

DFHLS2SC and DFHLS2WS do not fully implement the padding algorithms of PL/I; therefore, you must declare padding bytes explicitly in your data structure. DFHLS2SC and DFHLS2WS issue a message if they detect that padding bytes are missing. Each top-level structure must start on a double-word boundary and each byte in 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 the fields in the following order:

1. FIELD3 is aligned first because it has the strongest boundary requirements.
2. FIELD2 is aligned at the fullword boundary immediately before FIELD3.
3. FIELD1 is aligned 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 that 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 (`*n*`)`
where *n* ≤ 7
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:byte"/>`
	`</xsd:simpleType>`

PL/I data description	Schema
FIXED BINARY (*n*) where 8 ≤ *n* ≤ 15	<xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType>
FIXED BINARY (*n*) where 16 ≤ *n* ≤ 31	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>
FIXED BINARY (*n*) where 32 ≤ *n* ≤ 63 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where *n* ≤ 8 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where 9 ≤ *n* ≤ 16 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where 17 ≤ *n* ≤ 32 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where 33 ≤ *n* ≤ 64 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType>
FIXED DECIMAL(*n*,*m*)	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>
FIXED DECIMAL(15) Supported at mapping level 3.0 and higher when DATETIME=PACKED15	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> The format of the time stamp is CICS ABSTIME.
BIT(*n*) where *n* is a multiple of 8. Other values are not supported.	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> where *m* = *n*/8
CHARACTER(*n*) VARYING and VARYINGZ are also supported at mapping level 1.2 and higher. **Restriction:** VARYINGZ is supported only by Enterprise PL/I	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>

PL/I data description	Schema
GRAPHIC(*n*) VARYING and VARYINGZ are also supported at mapping level 1.2 and higher. **Restriction:** VARYINGZ is supported only by Enterprise PL/I	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.0 and 1.1, where *m* = 2**n* <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.2 or higher
WIDECHAR(*n*) **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.0 and 1.1, where *m* = 2**n* <xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*n*"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.2 or higher
ORDINAL **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
BINARY FLOAT(*n*) where *n* <= 21 Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
BINARY FLOAT(*n*) where 21 < *n* <= 53 Values greater than 53 are not supported. Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>
DECIMAL FLOAT(*n*)where *n* <= 6 Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
DECIMAL FLOAT(*n*)where 6 < *n* <= 16 Values greater than 16 are not supported. Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

XML schema to PL/I mapping:

The DFHSC2LS and DFHWS2LS utility programs support mappings between XML schema definitions and PL/I data structures. Because the Enterprise PL/I compiler and older PL/I compilers differ, two language options are supported: `PLI-ENTERPRISE` and `PLI-OTHER`.

The CICS assistants generate 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` attributes 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 more 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` attributes are specified with different values.

   For more information, see “Variable arrays of elements” on page 112.
5. For attributes, the previous rules are applied to the element name. The prefix `attr-` is added to the element name and is followed by `-value` or `-exist`. If the total length is longer than 28 characters, the element name is truncated. For more information, see “Support for XML attributes” on page 116.

   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 31 characters or less.

DFHSC2LS and DFHWS2LS map schema types to PL/I data types according to the following table. 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 and an extra character is allocated for the null-terminator.
- 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 less than 32 768 bytes and to a container if it is more than 32 768 bytes.

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	Mapping level 2.0 and below: Not supported Mapping level 2.1 and higher: Supported
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and higher:CHAR(255)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <xsd:maxLength value="*z*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> where *type* is one of: string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY	All mapping levels:CHARACTER(*z*)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> where *type* is one of: duration date time gDay gMonth gYear gMonthDay gYearMonth	All mapping levels:CHAR(32)
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.2 and below: CHAR(32) Mapping level 2.0 and higher: CHAR(40) Mapping level 3.0 and higher: FIXED DECIMAL(15) The format of the time stamp is CICS ABSTIME.

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*y*"/> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: BIT(*z*) where *z* = 8 ×*y* and *z* < 4095 bytes. CHAR(*z*) where *z* = 8 ×*y* and *z* > 4095 bytes. Mapping levels 1.2 and higher: CHAR(*y*)
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** SIGNED FIXED BINARY (7) **Other PL/I** FIXED BINARY (7)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** UNSIGNED FIXED BINARY (8) **Other PL/I** FIXED BINARY (8)
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** SIGNED FIXED BINARY (15) **Other PL/I** FIXED BINARY (15)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** UNSIGNED FIXED BINARY (16) **Other PL/I** FIXED BINARY (16)
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** FIXED DECIMAL(31,0) **Other PL/I** FIXED DECIMAL(15,0)
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** SIGNED FIXED BINARY (31) **Other PL/I** FIXED BINARY (31)

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** UNSIGNED FIXED BINARY(32) Mapping level 1.2 and higher: **Enterprise PL/I** CHAR(*y*) where *y* is a fixed length that is less than 16 MB. All mapping levels: **Other PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** SIGNED FIXED BINARY(63) Mapping level 1.2 and higher: **Enterprise PL/I** CHAR(*y*) where *y* is a fixed length that is less than 16 MB. All mapping levels: **Other PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** UNSIGNED FIXED BINARY(64) Mapping level 1.2 and higher: **Enterprise PL/I** CHAR(*y*) where *y* is a fixed length that is less than 16 MB. All mapping levels: **Other PL/I** BIT(64)

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** SIGNED FIXED BINARY (7) **Other PL/I** FIXED BINARY (7) Mapping level 1.2 and higher: **Enterprise PL/I** BIT(7) BIT(1) **Other PL/I** BIT(7) BIT(1) where BIT(7) is provided for alignment and BIT(1) contains the Boolean mapped value.
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>	All mapping levels:FIXED DECIMAL(*n*,*m*)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	All mapping levels:CHAR(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	All mapping levels:CHAR(255)
<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> where the length is not defined	Mapping level 1.0: Not supported Mapping level 1.1: CHAR(*z*) where *z* =4×(ceil(*y*/3)). ceil(*x*) is the smallest integer greater than or equal to *x*. Mapping level 1.2 and higher: CHAR(*y*) where the length is fixed. CHAR(16) where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	Mapping levels 1.0 and 1.1: CHAR(32) Mapping level 1.2 and higher: **Enterprise PL/I** DECIMAL FLOAT(6) HEXADEC **Other PL/I** DECIMAL FLOAT(6)
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	Mapping levels 1.0 and 1.1: CHAR(32) Mapping level 1.2 and higher: **Enterprise PL/I** DECIMAL FLOAT(16) HEXADEC **Other PL/I** DECIMAL FLOAT(16)

Variable arrays of elements:

XML can contain an array with varying numbers of elements. In general, WSDL documents and XML schemas that contain varying numbers of elements do not map efficiently into a single high-level language data structure. CICS uses container-based mappings or inline mappings to handle varying numbers of elements in XML.

An array with a varying number of elements is represented in the XML schema by using the `minOccurs` and `maxOccurs` attributes on the element declaration:

- The `minOccurs` attribute specifies the minimum number of times that the element can occur. It can have a value of 0 or any positive integer.
- The `maxOccurs` attribute specifies the maximum number of times that 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 no upper limit applies to the number of times the element can occur.
- The default value for both attributes is 1.

This example denotes an 8-byte string that is optional; that is, it can occur never or once in the application XML or SOAP message:

```
<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>
```

The following example denotes an 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, WSDL documents 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 transforms XML to application data, it populates these structures with the application data and the application reads them.
- When CICS transforms the application data to XML, it reads the application data in the structures that have been populated by the application.

The format of these data structures is best explained with a series of examples. The XML can be from a SOAP message or from an application. 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
```

Varying number of elements at mapping level 2 and below

This 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 main data structure contains a declaration of two fields. When CICS transforms the XML to binary data, the first field `component-num` contains the number of times that the element appears in the XML, and the second field, `component-cont`, contains the name of a container:

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

A second data structure contains the declaration of the element itself:

```
01 DFHWS-component
  02 component PIC X(8)
```

You must examine the value of `component-num` (which will contain a value in the range 1 to 5) to find out how many times the element occurs. The element contents

are in the container named in component-cont; the container holds an array of elements, where each element is mapped by the DFHWS-component data structure.

If minOccurs="0" and maxOccurs="1", the element is optional. To process the data structure in your application program, you must examine the value of component-num:

- If it is zero, the message has no component element 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.

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.

Varying number of elements at mapping level 2.1 and above

At mapping level 2.1 and above, you can use the **INLINE-MAXOCCURS-LIMIT** parameter in the CICS assistants. The **INLINE-MAXOCCURS-LIMIT** parameter specifies the way that varying numbers of elements are handled. The mapping options for varying numbers of elements are container-based mapping, described in "Varying number of elements at mapping level 2 and below" on page 113, or inline mapping. The *value* of this parameter can be a positive integer in the range 0 - 32767:

- The default value of **INLINE-MAXOCCURS-LIMIT** is 1, which ensures that optional elements are mapped inline.
- A value of 0 for the **INLINE-MAXOCCURS-LIMIT** parameter prevents inline mapping.
- If maxOccurs is less than or equal to the value of **INLINE-MAXOCCURS-LIMIT**, inline mapping is used.
- If maxOccurs is greater than the value of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used.

Mapping varying numbers of elements inline results in the generation of both an array, as happens with the fixed occurrence example above, and a counter. The component-num field indicates how many instances of the element are present, and these are pointed to by the array. For the example shown in "Varying number of elements at mapping level 2 and below" on page 113, when **INLINE-MAXOCCURS-LIMIT** is less than or equal to 5, the generated data structure is like this:

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

The first field, component-num, is identical to the output for the container-based mapping example in the previous section. The second field contains an array of length 5 which is large enough to contain the maximum number of elements that can be generated.

Inline mapping differs from container-based mapping, which stores the number of occurrences of the element and the name of the container where the data is placed, because it stores all the data in the current container. Storing the data in the current container will generally improve performance and make inline mapping preferable.

Nested variable arrays

Complex WSDL documents and XML schemas can contain variably recurring elements, which in turn contain variably recurring elements. In this case, the structure described extends beyond the two levels described in the examples.

This example illustrates an optional element called <component2> that is nested in a mandatory element called <component1>, where the mandatory element 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-5
05 component1-cont PIC X(16)
```

However, the second data structure contains these elements:

```
01 DFHWS-component1
  02 component2-num PIC S9(9) COMP-5
  02 component2-cont PIC X(16)
```

A third-level structure contains these elements:

```
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 structure, consider the fragment of XML that matches the example:

```
<component1><component2>string1</component2></component1>
<component1><component2>string2</component2></component1>
<component1></component1>
```

<component1> occurs three times. 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. The container named in component1-cont has 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 are 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

Optional structures and xsd:choice

DFHWS2LS and DFHSC2LS support the use of maxOccurs and minOccurs on <xsd:sequence>, <xsd:choice>, and <xsd:all> elements only at mapping level 2.1 and above, where the minOccurs and maxOccurs attributes are set to minOccurs="0" and maxOccurs="1".

The assistants generate mappings that treat these elements as though each child element in them is optional. When you implement an application with these elements, ensure that invalid combinations of options are not generated by the application. Each of the elements has its own count field in the generated languages structure, these fields must either all be set to "0" or all be set to"1". Any other combination of values is invalid, except for with <xsd:choice> elements.

<xsd:choice> elements indicate that only one of the options in the element can be used. It is supported at all mapping levels. The assistants handle each of the options in an <xsd:choice> as though it is in an <xsd:sequence> element with minOccurs="0" and maxOccurs="1". Take care when you implement an application using the <xsd:choice> element to ensure that invalid combinations of options are not generated by the application. Each of the elements has its own count field in the generated languages structure, exactly one of which must be set to '1' and the others must all be set to '0'. Any other combination of values is invalid, except when the <xsd:choice> element is itself optional, in which case it is valid for all the fields to be set to '0'.

Support for XML attributes:

XML schemas can specify attributes that are allowed or required in XML. The CICS assistant utilities DFHWS2LS and DFHSC2LS ignore XML attributes by default. To process XML attributes that are defined in the XML schema, the value of the **MAPPING-LEVEL** parameter must be 1.1 or higher.

Optional attributes

Attributes can be optional or required and can be associated with any element in a SOAP message or XML for an application. 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 1 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 with which they are associated. 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 maps 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

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 transforms the data to XML, the following runtime processing takes place:

- If the existence flag is set, the attribute is transformed and included in the XML.
- If the existence flag is not set, the attribute is not included in the XML.

Required attributes and runtime processing

For every attribute that is required, only the value field is generated in the appropriate language structure.

If the attribute is present in the XML, the value is mapped. If the attribute is not present, the following processing occurs:

- If the application is a Web service provider, CICS generates a SOAP fault message indicating an error in the client SOAP message.
- If the application is a Web service requester, CICS issues a message and returns a conversion error response with a RESP2 code of 13 to the application.
- If the application is using the **TRANSFORM XMLTODATA** command, CICS issues a message and returns an invalid request response with a RESP2 code of 3 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. When an application uses the **TRANSFORM DATATOXML** command, CICS also transforms the attribute and includes it in the XML.

The nillable attribute

The nillable attribute is a special attribute that can appear on an `xsd:element` in an XML schema. It specifies that the `xsi:nil` attribute is valid for the element in XML. 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, it is mapped as a required attribute that takes a Boolean value.

When CICS receives a SOAP message or has to transform XML for an application that contains an `xsi:nil` attribute, the value of the attribute is true or false. If the value is true, the application must ignore the values of the element or nested elements in the scope of the `xsi:nil` attribute.

When CICS produces a SOAP message or XML based on the contents of a COMMAREA or container for which the value for the `xsi:nil` attribute is true, the following processing occurs:

- The `xsi:nil` attribute is generated into the XML or SOAP message.
- The value of the associated element is ignored.
- Any nested elements within the element are ignored.

SOAP message example

Consider the following example XML schema, which could be part of a WSDL document:

```
<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>
```

Here is an example of a partial SOAP message that conforms to this schema:

```
<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 maps to these elements:

```
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)
```

Support for <xsd:any> and `xsd:anyType`:

DFHWS2LS and DFHSC2LS support the use of <xsd:any> and `xsd:anyType` in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. `xsd:anyType` is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

Before you can use <xsd:any> and `xsd:anyType` with the CICS assistants, set the following parameters:

- Set the **`MAPPING-LEVEL`** parameter to `2.1` or higher.
- For a web service provider application, set the **`PGMINT`** parameter to CHANNEL.

<xsd:any> example

This example uses an <xsd:any> element to describe some optional unstructured XML content following the `"Surname"` tag in the `"Customer"` tag:

```
<xsd:element name="Customer">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="FirstName" type="xsd:string"/>
   <xsd:element name="Surname" type="xsd:string"/>
   <xsd:any minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

An example SOAP message that conforms to this XML schema is:

```
<xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Customer xmlns="http://www.example.org/anyExample">
      <Title xmlns="">Mr</Title>
      <FirstName xmlns="">John</FirstName>
      <Surname xmlns="">Smith</Surname>
      <ExtraInformation xmlns="http://www.example.org/ExtraInformation">
        <!-- This 'ExtraInformation' tag is associated with the optional xsd:any from the XML schema.
             It can contain any well formed XML. -->
        <ExampleField1>one</ExampleField1>
        <ExampleField2>two</ExampleField2>
      </ExtraInformation>
    </Customer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If this SOAP message is sent to CICS, CICS populates the `Customer-xml-cont` container with the following XML data:

```
<ExtraInformation xmlns="http://www.example.org/ExtraInformation">
  <!-- This 'ExtraInformation' tag is associated with the optional xsd:any from the XML schema.
       It can contain any well formed XML. -->
  <ExampleField1>one</ExampleField1>
  <ExampleField2>two</ExampleField2>
</ExtraInformation>
```

CICS also populates the `Customer-xmlns-cont` container with the following XML namespace declarations that are in scope; these declarations are separated by a space:

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns="http://www.example.org/anyExample"
```

xsd:anyType example

The `xsd:anyType` is the base data type from which all simple and complex data types are derived. It does not restrict the data content. If you do not specify a data type, it defaults to `xsd:anyType`; for example, these two XML fragments are equivalent:

```
<xsd:element name="Name" type="xsd:anyType"/>
```

```
<xsd:element name="Name"/>
```

Generated language structures

The language structures generated for <xsd:any> or `xsd:anyType` take the following form in COBOL and an equivalent form for the other languages:

elementName-xml-cont PIC X(16)
: The name of a container that holds the raw XML. When CICS processes an incoming SOAP message, it places the subset of the SOAP message that the <xsd:any> or `xsd:anyType` defines into this container. The application can

process the XML data only natively. The application must generate the XML, populate this container, and supply the container name.

This container must be populated in text mode. If CICS populates this container, it does so using the same variant of EBCDIC as the web service is defined to use. Characters that do not exist in the target EBCDIC code page are replaced with substitute characters, even if the container is read by the application in UTF-8.

elementName-xmlns-cont PIC X(16)
The name of a container that holds any namespace prefix declarations that are in scope. The contents of this container are similar to those of the DFHWS-XMLNS container, except that it includes all the namespace declarations that are in scope and that are relevant, rather than only the subset from the SOAP Envelope tag.

This container must be populated in text mode. If CICS populates this container, it does so using the same variant of EBCDIC as the web service is defined to use. Characters that do not exist in the target EBCDIC code page are replaced with substitute characters, even if the container is read by the application in UTF-8.

This container is used only when processing SOAP messages sent to CICS. If the application tries to supply a container with namespace declarations when an output SOAP message is generated, the container and its contents are ignored by CICS. CICS requires that the XML supplied by the application is entirely self-contained with respect to namespace declarations.

The name of the XML element that contains the <xsd:any> element is included in the variable names that are generated for the <xsd:any> element. In the <xsd:any> example, the <xsd:any> element is nested inside the <xsd:element name="Customer"> element and the variable names that are generated for the <xsd:any> element are `Customer-xml-cont PIC X(16)` and `Customer-xmlns-cont PIC X(16)`.

For an `xsd:anyType` type, the direct XML element name is used; in the `xsd:anyType` example above, the variable names are `Name-xml-cont PIC X(16)` and `Name-xmlns-cont PIC X(16)`.

Support for <xsd:choice>:

An `<xsd:choice>` element indicates that only one of the options in the element can be used. The CICS assistants provide varying degrees of support for `<xsd:choice>` elements at the various mapping levels.

Support for <xsd:choice> at mapping level 2.2 and higher

At mapping level 2.2 and higher, DFHWS2LS and DFHSC2LS provide improved support for `<xsd:choice>` elements. The assistants generate a new container that stores the value associated with the `<xsd:choice>` element. The assistants generate language structures containing the name of a new container and an extra field:

fieldname-**enum**
The discriminating field to indicate which of the options the `<xsd:choice>` element will use.

fieldname-**cont**
The name of the container that stores the option to be used. A further language structure is generated to map the value of the option.

The following XML schema fragment includes an <xsd:choice> element:

```
<xsd:element name="choiceExample">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element name="option1" type="xsd:string" />
         <xsd:element name="option2" type="xsd:int" />
         <xsd:element name="option3" type="xsd:short" maxOccurs="2" minOccurs="2" />
      </xsd:choice>
   </xsd:complexType>
</xsd:element>
```

If this XML schema fragment is processed at mapping level 2.2 or higher, the assistant generates the following COBOL language structures:

```
   03 choiceExample.
      06 choiceExample-enum         PIC X DISPLAY.
          88 empty                     VALUE X'00'.
          88 option1                   VALUE X'01'.
          88 option2                   VALUE X'02'.
          88 option3                   VALUE X'03'.
      06 choiceExample-cont         PIC X(16).


01 Example-option1.
   03 option1-length                PIC S9999 COMP-5 SYNC.
   03 option1                       PIC X(255).

01 Example-option2.
   03 option2                       PIC S9(9) COMP-5 SYNC.

01 Example-option3.
   03 option3 OCCURS 2             PIC S9999 COMP-5 SYNC.
```

Limitations for <xsd:choice> at mapping level 2.2 and higher

DFHSC2LS and DFHWS2LS do not support nested <xsd:choice> elements; for example, the following XML is not supported:

```
<xsd:choice>
   <xsd:element name ="name1" type="string"/>
   <xsd:choice>
      <xsd:element name ="name2a" type="string"/>
      <xsd:element name ="name2b" type="string"/>
   </xsd:choice>
</xsd:choice>
```

DFHSC2LS and DFHWS2LS do not support recurring <xsd:choice> elements; for example, the following XML is not supported:

```
<xsd:choice maxOccurs="2">
   <xsd:element name ="name1" type="string"/>
</xsd:choice>
```

DFHSC2LS and DFHWS2LS support a maximum of 255 options in an <xsd:choice> element.

Support for <xsd:choice> at mapping level 2.1 and below

At mapping level 2.1 and below, DFHWS2LS provides limited support for <xsd:choice> elements. DFHWS2LS treats each of the options in an <xsd:choice> element as though it is an <xsd:sequence> element that can occur at most once.

Only one of the options in an <xsd:choice> element can be used, so take care when you implement an application using the <xsd:choice> element that you generate only valid combinations of options. Each of the elements has its own count field in the generated languages structure, exactly one of which must be set to 1 and the others must all be set to 0. Any other combination of values is incorrect, except when the <xsd:choice> is itself optional, in which case it is valid for all of the fields to be set to 0.

Related reference:

"Support for <xsd:any> and xsd:anyType" on page 118
DFHWS2LS and DFHSC2LS support the use of <xsd:any> and xsd:anyType in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. xsd:anyType is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

"Support for abstract elements and abstract data types" on page 123
The CICS assistants provide support for abstract elements and abstract data types at mapping level 2.2 and higher. The CICS assistants map abstract elements and abstract data types in a similar way to substitution groups.

"Support for substitution groups"
You can use a substitution group to define a group of XML elements that are interchangeable. The CICS assistants provide support for substitution groups at mapping level 2.2 and higher.

Support for substitution groups:

You can use a substitution group to define a group of XML elements that are interchangeable. The CICS assistants provide support for substitution groups at mapping level 2.2 and higher.

At mapping level 2.2 and higher, DFHSC2LS and DFHWS2LS support substitution groups using similar mappings to those used for <xsd:choice> elements. The assistant generates an enumeration field and a new container name in the language structure.

The following XML schema fragment includes an array of two subGroupParent elements, each of which can be replaced with replacementOption1 or replacementOption2:

```
<xsd:element name="subGroupExample" >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="subGroupParent" maxOccurs="2" minOccurs="2" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="subGroupParent" type="xsd:anySimpleType" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="subGroupParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="subGroupParent" />
```

Processing this XML fragment with the assistant generates the following COBOL language structures:

```
   03 subGroupExample.
      06 subGroupParent OCCURS2.
         09 subGroupExample-enum PIC X DISPLAY.
            88 empty              VALUE X '00'.
            88 replacementOption1  VALUE X '01'.
            88 replacementOption2  VALUE X '02'.
            88 subGroupParent      VALUE X '03'.
         09 subGroupExample-cont PIC X (16).



01 Example-replacementOption1.
   03 replacementOption1      PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
   03 replacementOption2      PIC S9999 COMP-5 SYNC.

01 Example-subGroupParent.
   03 subGroupParent-length   PIC S9999 COMP-5 SYNC.
   03 subGroupParent          PIC X(255).
```

For more information about substitution groups, see the *W3C XML Schema Part 1: Structures Second Edition specification*: http://www.w3.org/TR/xmlschema-1/#Elements_Equivalence_Class

Related reference:

"Support for <xsd:any> and xsd:anyType" on page 118
DFHWS2LS and DFHSC2LS support the use of <xsd:any> and xsd:anyType in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. xsd:anyType is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

"Support for <xsd:choice>" on page 120
An <xsd:choice> element indicates that only one of the options in the element can be used. The CICS assistants provide varying degrees of support for <xsd:choice> elements at the various mapping levels.

"Support for abstract elements and abstract data types"
The CICS assistants provide support for abstract elements and abstract data types at mapping level 2.2 and higher. The CICS assistants map abstract elements and abstract data types in a similar way to substitution groups.

Support for abstract elements and abstract data types:

The CICS assistants provide support for abstract elements and abstract data types at mapping level 2.2 and higher. The CICS assistants map abstract elements and abstract data types in a similar way to substitution groups.

Support for abstract elements at mapping level 2.2 and higher

At mapping level 2.2 and above, DFHSC2LS and DFHWS2LS treat abstract elements in almost the same way as substitution groups except that the abstract element is not a valid member of the group. If there are no substitutable elements, the abstract element is treated as an <xsd:any> element and uses the same mappings as an <xsd:any> element at mapping level 2.1.

The following XML schema fragment specifies two options that can be used in place of the abstract element. The abstract element itself is not a valid option:

```
<xsd:element name="abstractElementExample" >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="abstractElementParent" maxOccurs="2" minOccurs="2" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="abstractElementParent" type="xsd:anySimpleType" abstract="true" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="abstractElementParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="abstractElementParent" />
```

Processing this XML fragment with the assistant generates the following COBOL language structures:

```
   03 abstractElementExample.
      06 abstractElementParent OCCURS 2.
         09 abstractElementExample-enum  PIC X DISPLAY.
            88 empty                          VALUE X '00'.
            88 replacementOption1             VALUE X '01'.
            88 replacementOption2             VALUE X '02'.
         09 abstractElementExample-cont  PIC X (16).


01 Example-replacementOption1.
   03 replacementOption1                  PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
   03 replacementOption2                  PIC S9999 COMP-5 SYNC.
```

For more information about abstract elements, see the *W3C XML Schema Part 0: Primer Second Edition specification*: http://www.w3.org/TR/xmlschema-0/#SubsGroups

Support for abstract data types at mapping level 2.2 and higher

At mapping level 2.2 and higher, DFHSC2LS and DFHWS2LS treat abstract data types as substitution groups. The assistant generates an enumeration field and a new container name in the language structure.

The following XML schema fragment specifies two alternatives that can be used in place of the abstract type:

```
<xsd:element name="AbstractDataTypeExample" type="abstractDataType" />

<xsd:complexType name="abstractDataType" abstract="true">
   <xsd:simpleContent>
      <xsd:extension base="xsd:string" />
   </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option1">
   <xsd:simpleContent>
      <xsd:restriction base="abstractDataType">
         <xsd:length value="5" />
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option2">
   <xsd:simpleContent>
      <xsd:restriction base="abstractDataType">
         <xsd:length value="10" />
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
```

Processing this XML fragment with the assistant generates the following COBOL language structures:

```
   03 AbstractDataTypeExamp-enum   PIC X DISPLAY.
     88 empty                         VALUE X'00'.
     88 option1                       VALUE X'01'.
     88 option2                       VALUE X'02'.
   03 AbstractDataTypeExamp-cont   PIC X(16).
```

The language structures are generated into separate copy books. The language structure generated for `option1` is generated into one copybook:

```
   03 option1          PIC X(5).
```

The language structure for `option2` is generated into a different copybook:

```
   03 option2          PIC X(10).
```

For more information about abstract data types, see the *W3C XML Schema Part 0: Primer Second Edition specification*: http://www.w3.org/TR/xmlschema-0/#SubsGroups

Related reference:

"Support for <xsd:any> and `xsd:anyType`" on page 118
DFHWS2LS and DFHSC2LS support the use of <xsd:any> and `xsd:anyType` in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. `xsd:anyType` is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

"Support for `<xsd:choice>`" on page 120
An `<xsd:choice>` element indicates that only one of the options in the element can be used. The CICS assistants provide varying degrees of support for `<xsd:choice>` elements at the various mapping levels.

"Support for substitution groups" on page 122
You can use a substitution group to define a group of XML elements that are interchangeable. The CICS assistants provide support for substitution groups at mapping level 2.2 and higher.

How to handle variably repeating content in COBOL:

In COBOL, you cannot process variably repeating content by using pointer arithmetic to address each instance of the data. Other programming languages do not have this limitation. This example shows you how to handle variably repeating content in COBOL for a web service application.

This technique also applies to transforming XML to application data using the **TRANSFORM** API commands. The following example WSDL document represents a web service with application data that consists of an 8-character string that recurs a variable number of times:

```
<?xml version="1.0"?>
<definitions name="ExampleWSDL"
             targetNamespace="http://www.example.org/variablyRepeatingData/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://www.example.org/variablyRepeatingData/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <types>
   <xsd:schema targetNamespace="http://www.example.org/variablyRepeatingData/">
     <xsd:element name="applicationData">
       <xsd:complexType>
         <xsd:sequence>
```

```
          <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>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
 </types>

 <message name="exampleMessage">
   <part element="tns:applicationData" name="messagePart"/>
 </message>

 <portType name="examplePortType">
   <operation name="exampleOperation">
     <input message="tns:exampleMessage"/>
     <output message="tns:exampleMessage"/>
   </operation>
 </portType>

 <binding name="exampleBinding" type="tns:examplePortType">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="exampleOperation">
     <soap:operation soapAction=""/>
     <input><soap:body parts="messagePart" encodingStyle="" use="literal"/></input>
     <output><soap:body parts="messagePart" encodingStyle="" use="literal"/></output>
   </operation>
 </binding>
</definitions>
```

Processing this WSDL document through DFHWS2LS generates the following COBOL language structures:

```
          03 applicationData.

            06 component-num                PIC S9(9) COMP-5 SYNC.
            06 component-cont               PIC X(16).


        01 DFHWS-component.
          03 component                     PIC X(8).
```

Note that the 8-character `component` field is defined in a separate structure called `DFHWS-component`. The main data structure is called `applicationData` and it contains two fields, `component-num` and `component-cont`. The `component-num` field indicates how many instances of the `component` data are present and the `component-cont` field indicates the name of a container that holds the concatenated list of `component` fields.

The following COBOL code demonstrates one way to process the list of variably recurring data. It makes use of a linkage section array to address subsequent instances of the data, each of which is displayed by using the `DISPLAY` statement:

```
IDENTIFICATION DIVISION.
       PROGRAM-ID.    EXVARY.

       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      * working storage variables
       01 APP-DATA-PTR             USAGE IS POINTER.
       01 APP-DATA-LENGTH          PIC S9(8) COMP.
```

```
 01 COMPONENT-PTR            USAGE IS POINTER.
 01 COMPONENT-DATA-LENGTH    PIC S9(8) COMP.
 01 COMPONENT-COUNT          PIC S9(8) COMP-4 VALUE 0.
 01 COMPONENT-LENGTH         PIC S9(8) COMP.

 LINKAGE SECTION.

* a large linkage section array
 01 BIG-ARRAY PIC X(659999).

* application data structures produced by DFHWS2LS
* this is normally referenced with a COPY statement
 01 DFHWS2LS-data.
    03 applicationData.
      06 component-num  PIC S9(9) COMP-5 SYNC.
      06 component-cont PIC X(16).

 01 DFHWS-component.
    03 component        PIC X(8).



 PROCEDURE DIVISION USING DFHEIBLK.
 A-CONTROL SECTION.
 A010-CONTROL.

* Get the DFHWS-DATA container
     EXEC CICS GET CONTAINER('DFHWS-DATA')
               SET(APP-DATA-PTR)
               FLENGTH(APP-DATA-LENGTH)
     END-EXEC
     SET ADDRESS OF DFHWS2LS-data TO APP-DATA-PTR

* Get the recurring component data
     EXEC CICS GET CONTAINER(component-cont)
               SET(COMPONENT-PTR)
               FLENGTH(COMPONENT-DATA-LENGTH)
     END-EXEC

* Point the component structure at the first instance of the data
     SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* Store the length of a single component
     MOVE LENGTH OF DFHWS-component TO COMPONENT-LENGTH

* process each instance of component data in turn
     PERFORM WITH TEST AFTER
           UNTIL COMPONENT-COUNT = component-num

* display the current instance of the data
        DISPLAY 'component value is: ' component

* address the next instance of the component data
        SET ADDRESS OF BIG-ARRAY TO ADDRESS OF DFHWS-component
        SET ADDRESS OF DFHWS-component
            TO ADDRESS OF BIG-ARRAY (COMPONENT-LENGTH + 1:1)
        ADD 1 TO COMPONENT-COUNT

* end the loop
     END-PERFORM.

* Point the component structure back at the first instance of
* of the data, for any further processing we may want to perform
     SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* return to CICS.
```

```
           EXEC CICS
                RETURN
           END-EXEC

           GOBACK.
```

The code above provides a generic solution to handling variably repeating content. The array, BIG-ARRAY, moves to the start of each component in turn and does not remain fixed at the start of the data. The component data structure is then moved to point at the first byte of the next component. COMPONENT-PTR can be used to recover the start position of the component data if required.

Here is an example SOAP message that conforms to the WSDL document:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
   <applicationData xmlns="http://www.example.org/variablyRepeatingData/">
     <component xmlns="">VALUE1</component>
     <component xmlns="">VALUE2</component>
     <component xmlns="">VALUE3</component>
   </applicationData>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here is the output produced by the COBOL program when it processes the SOAP message:

```
CPIH 20080115103151 component value is: VALUE1
CPIH 20080115103151 component value is: VALUE2
CPIH 20080115103151 component value is: VALUE3
```

Creating a web service provider by using the web services assistant

You can create a service provider application from a web service description that complies with WSDL 1.1 or WSDL 2.0, or from a high-level language data structure. The CICS web services assistant helps you to deploy your CICS applications in a service provider setting.

About this task

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 interfaces in the web service description and the SOAP messages.

You can expose the web service description associated with your service provider using a URI. This URI has the same path as the URI associated with the WEBSERVICE with the suffix ?wsdl appended. This enables requesters within your business, or external to it, to discover the WSDL files associated with your service providers.

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 that complies with WSDL 1.1 or WSDL 2.0.

Before you begin

Before you can create a service provider application, the following conditions must be satisfied:

- Your web services description must be in a UNIX file in z/OS and you must create a suitable provider mode pipeline in the CICS region.
- You must define to OMVS the user ID under which DFHWS2LS runs.
- The user ID must have read permission to z/OS UNIX and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **WSDL** parameters.
- You must allocate sufficient storage to the user ID for the ID to run Java. You can use any supported version of Java. By default, DFHWS2LS uses the Java version specified in the **JAVADIR** parameter.

About this task

You can use the web services assistant to create language structures from your WSDL for the service provider application. You can also use a WSDL document that is stored in an IBM webSphere Service Registry and Repository (WSRR) server.

Procedure

1. Use the DFHWS2LS batch program to generate a web service binding file and one or more language data structures. DFHWS2LS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider these options when you enable an existing application for web services:
   a. **Which mechanism will CICS use to pass data to the service provider application program?**
      - You can use channels and pass the data in containers or use a COMMAREA. Channels and containers are recommended. Specify them with the **PGMINT** parameter.
   b. **Which language do you want to generate?**
      - DFHWS2LS can generate COBOL, C/C++, or PL/I language data structures. Specify the language using the **LANG** parameter.
   c. **Which mapping level do you want to use?**
      - The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to use the highest level of mapping available. Specify the mapping level with the **MAPPING-LEVEL** parameter.
   d. **Which URI do you want the web service requester to use?**
      - Specify a relative URI using the **URI** parameter; for example, `URI=/my/test/webservice`. The value is used by CICS when it creates the URIMAP resource.
   e. **Under which transaction and user ID will you run the web service request and response?**

- You can use an alias transaction to run the application to compose a response to the service requester. The alias transaction is attached under the user ID.
- Specify it with the **TRANSACTION** and **USERID** parameters. These values are used when creating the URIMAP resource. If you do not want to use a specific transaction, do not use these parameters.

f. **Where is the WSDL document stored?**
- If you want to retrieve a WSDL document from a WSRR server, instead of from the local file system, you must specify certain parameters in DFHWS2LS.
- As a minimum, you must specify the **WSRR-SERVER** parameter with the location of the WSRR server and the **WSRR-NAME** parameter with the name of the WSDL document that you want to retrieve from WSRR.
- For information about other parameters that you might want to specify if you are using WSRR, see "DFHWS2LS: WSDL to high-level language conversion" on page 66.

g. **If you intend to retrieve your WSDL document from a WSRR server, do you want to do so using a secure connection?**
- You can use secure socket layer (SSL) encryption by setting the appropriate parameters to interoperate securely with WSRR. For an example, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

- When you submit DFHWS2LS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The language structures are placed in the partitioned data set that you specified with the **PDSLIB** parameter.

2. Copy the generated web service binding file to the `pickup` directory of the provider mode PIPELINE resource that you want to use for your web service application. You must copy the binding file in binary mode.
3. Optional: Copy the web service description or the archive file containing one or more web service descriptions to the same directory as the web service binding file. The archive file must be a .zip file and the file name must match the WSDL file name. With this copy, you can discover the WSDL.
4. Write a service provider application program to interface with the generated language structures and implement the required business logic.
5. Create the WEBSERVICE resource and two URIMAP resources.
   - The WEBSERVICE resource encapsulates the web service binding file in CICS and is used at run time.
   - The first URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.
   - The second URIMAP resource provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI.
     - This URI has the same path as the URI associated with the WEBSERVICE with the suffix `?wsdl` appended.
     - This URIMAP resource is created so that external requesters can use the URI to discover the WSDL archive file or WSDL document.
     - This URIMAP resource is created only if the web service description or the archive file containing one or more web service descriptions has been copied to the same directory as the web service binding file.
     - If the pickup directory contains a WSDL archive file and a WSDL document, the URI returns only the WSDL in the archive file.

- This function is only available for web services installed using the pipeline scan operation.

You can create the resources in the following ways:

a. Using the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and URIMAP resources.
b. Defining the resources yourself. If you use the CICS Explorer® to define a WEBSERVICE resource in a CICS bundle, you can choose to import a web service binding file and a WSDL document or WSDL archive file and include these in the bundle. You can then generate the URIMAP definitions to support the web service and package these in a bundle. For more help with using the CICS Explorer to create and edit resources in CICS bundles, seeWorking with bundles in the CICS Explorer product documentation.

Results

If you have any problems submitting DFHWS2LS, or the resources do not install correctly, see "Diagnosing deployment errors" on page 593.

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 begin

Before you create a service provider application, make sure that these preconditions have been completed:

- Your high-level language data structures must meet the following criteria:
  - The data structures must be defined separately from the source program; for example, in a COBOL copybook.
  - If your PL/I or COBOL application program uses different data structures for input and output, the data structures must be defined in two different members in a partitioned data set. If the same structure is used for input and output, the structure must be defined in a single member.

    For C and C++, your data structures can be in the same member in a partitioned data set.
- The data structures you process depend on whether you are using a wrapper program:
  - If you are using a wrapper program, the copybook is the interface to the wrapper program.
  - If you are not using a wrapper program, the copybook is the interface to the business logic.
- The language structures must be available in a partitioned data set and you must create a suitable PIPELINE resource in the CICS region:
  - You must define to OMVS the user ID under which DFHLS2WS runs.
  - The user ID must have read permission to z/OS UNIX 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. You can use any supported version of Java. By default, DFHLS2WS uses the Java version specified in the **JAVADIR** parameter.

Procedure

Follow these steps to create a service provider application from a data structure:

1. If the service provider application interface uses channels and many containers, create a channel description document that describes the interface in XML. You must put the channel description document in a suitable directory on z/OS UNIX. CICS uses this document to construct and deconstruct a SOAP message from the containers on a channel. Alternatively, you can use one container on a channel and not create a channel description document.
   - For more information on how to create a channel description document, see "Creating a channel description document" on page 134.
2. Use the DFHLS2WS batch program to generate a web service binding file and web service description from the language structure. DFHLS2WS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider these options when web service enabling an existing application:
   a. **Which mechanism will CICS use to pass data to the service provider application program?**
      - You can use channels and pass the data in containers or use a COMMAREA. Specify the mechanism using the **PGMINT** parameter. If your application interface uses channels and many containers, specify the **REQUEST-CHANNEL** parameter and optionally the **RESPONSE-CHANNEL**. You can only use these parameters when the mapping level is 3.0 or higher.
   b. **Which level of web service description (WSDL document) do you want to generate?**
      - CICS generates descriptions that comply with either WSDL 1.1 or WSDL 2.0 documents. If you want the service provider application to support requests that comply with both levels of WSDL, specify values for the **WSDL_1.1** and **WSDL_2.0** parameters. Ensure that the file names are different when using more than one WSDL parameter. This specification produces two web service descriptions and a binding file.
   c. **Which version of the SOAP protocol do you want to use?**
      - You can specify the version with the **SOAPVER** parameter. You are recommended to use the `ALL` value, which gives the flexibility to use either SOAP 1.1 or SOAP 1.2 as the binding for the web service description, although you must install the web service into a pipeline that is configured with the SOAP 1.2 message handler. You can use this parameter only when the **MINIMUM-RUNTIME-LEVEL** is 2.0 or higher.
   d. **Which mapping level do you want to use?**
      - The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to specify the highest level of mapping available in the **MAPPING-LEVEL** parameter.
   e. **Which URI do you want the web service requester to use?**
      - Specify an absolute URI using the **URI** parameter; for example, **URI**=`http://www.example.org:80/my/test/webservice`. The relative part of this address, `/my/test/webservice`, is used when creating the URIMAP resource. The full URI is used as the<soap:address> element in the web service description. This usage is true for both HTTP and WebSphere MQ URIs.

f. **Do you want to publish your WSDL document to an IBM WebSphere Service Registry and Repository (WSRR)?**
   - If you want to publish your WSDL document to a WSRR, you must specify the **WSRR-SERVER** parameter in DFHLS2WS. For more information on the parameters that you can specify when using WSRR, see "DFHLS2WS: high-level language to WSDL conversion" on page 53.

g. **If you intend to publish your WSDL document on a WSRR server, do you want to do so using a secure connection?**
   - You can use secure socket layer (SSL) encryption by setting the appropriate parameters to interoperate securely with WSRR. For an example, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

- When you submit DFHLS2WS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The generated web service description is placed in the location that you specified with the **WSDL**, **WSDL_1.1**, or **WSDL_2.0** parameter.
- If you have used the WSRR parameters in DFHLS2WS, your WSDL document is published to the WSRR server that you specified.

3. Review the generated web service description and perform any necessary customization. For more information, see "Customizing generated web service description documents" on page 135.
4. Copy the web service binding file to the pickup directory of the provider mode pipeline that you want to use for your web service application. You must copy the web service binding file in binary mode.
5. Optional: Copy the web service description or the archive file containing one or more web service descriptions to the same directory as the web service binding file. The archive file must be a .zip file and the file name must match the WSDL file name. With this copy, you can discover the WSDL.
6. Use the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and two URIMAP resources. The WEBSERVICE resource encapsulates the web service binding file in CICS and is used at run time.
   - The first URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.
   - The second URIMAP resource provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI.
     - This URI has the same path as the URI associated with the WEBSERVICE with the suffix `?wsdl` appended.
     - This URIMAP resource is created so that external requesters can use the URI to discover the WSDL archive file or WSDL document.
     - This URIMAP resource is created only if the web service description or the archive file containing one or more web service descriptions has been copied to the same directory as the web service binding file.
     - If the pickup directory contains a WSDL archive file and a WSDL document, the URI returns only the WSDL in the archive file.
     - This function is only available for web services installed using the pipeline scan operation.

   Alternatively, you can define the resources yourself, although this is not recommended.

Results

When you have successfully created the CICS resources, the creation of your service provider application is complete.

If you have any problems submitting DFHLS2WS, or the resources do not install correctly, see "Diagnosing deployment errors" on page 593.

What to do next

Make the web services description available to anyone who needs to develop a web service requester that will access your service.

Creating a channel description document:

Create a channel description document when your service provider application uses a channel interface with many containers.

About this task

Use an XML editor to create the channel description document. The schema for the channel description is called `channel.xsd` and is in the `/usr/lpp/cicsts/cicsts52/schemas/channel` directory (where `/usr/lpp/cicsts/cicsts52` is the default install directory for CICS files).

Procedure

1. Create an XML document with a <channel> element and the CICS channel namespace:

   ```
   <channel name="myChannel" xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
   </channel>
   ```

2. Add a <container> element for every container that the application program interface uses on the channel. You must use name, type and use attributes to describe each container. The following example shows six containers with different attribute values:

   ```
   <container name="cont1" type="char" use="required"/>
   <container name="cont2" type="char" use="optional"/>
   <container name="cont3" type="bit" use="required"/>
   <container name="cont4" type="bit" use="optional"/>
   <container name="cont5" type="bit" use="required">
       <structure location="//HLQ.PDSNAME(MEMBER)"/>
   </container>
   <container name="cont6" type="bit" use="optional">
       <structure location="//HLQ.PDSNAME(MEMBER2)"/>
   </container>
   ```

   The structure element indicates that the content is defined in a language structure located in a partitioned data set member.

3. Save the XML document in z/OS UNIX.

Channel schema

The channel description document must conform to the following schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
   xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel" elementFormDefault="qualified">
   <element name="channel"> 1
      <complexType>
         <sequence>
```

```
            <element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0"> 2
               <complexType>
                  <sequence>
                     <element name="structure" minOccurs="0"> 3
                       <complexType>
                           <attribute name="location" type="string" use="required"/>
                           <attribute name="structure" type="string" use="optional"/>
                       </complexType>
                     </element>
                  </sequence>
                  <attribute name="name" type="tns:name16Type" use="required"/>
                  <attribute name="type" type="tns:typeType" use="required"/>
                  <attribute name="use" type="tns:useType" use="required"/>
               </complexType>
             </element>
          </sequence>
          <attribute name="name" type="tns:name16Type" use="optional" />
       </complexType>
   </element>
  <simpleType name="name16Type">
      <restriction base="string">
         <maxLength value="16"/>
      </restriction>
  </simpleType>
  <simpleType name="typeType">
      <restriction base="string">
        <enumeration value="char"/>
        <enumeration value="bit"/>
      </restriction>
  </simpleType>
  <simpleType name="useType">
     <restriction base="string">
        <enumeration value="required"/>
        <enumeration value="optional"/>
     </restriction>
  </simpleType>
</schema>
```

1. This element represents a CICS channel.
2. This element represents a CICS container within the channel.
3. A structure can only be used with 'bit' mode containers. The 'location' attribute indicates the location of a file that maps the contents of container. The 'structure' attribute may be used in C and C++ to indicate the name of structure.

What to do next

Run DFHLS2WS to create the mappings and WSDL document for the web service provider application. DFHLS2WS puts the mappings for the channel in the WSDL document in the order that the containers are specified in the channel description document.

Customizing generated web service description documents:

The web service description (WSDL) documents that are generated by DFHLS2WS contain some automatically generated content that might be appropriate for you to change before publishing. Customizing WSDL documents can result in regenerating the web services binding file and, in some cases, writing a wrapper program.

About this task

Follow these steps to customize generated web service description documents:

Procedure

1. If you want to advertise support for HTTPS or communicate using WebSphere MQ, use the **URI** parameter in DFHLS2WS to set an absolute URI. If you have not used the **URI** parameter, you must change the <wsdl:service> and <wsdl:binding> elements 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, use the **URI** parameter in DFHLS2WS to set an absolute URI. If you have not used the **URI** parameter, add the details to the `soap:address` in 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 TCPIPSERVICE resource.
   b. If you are using WebSphere MQ as the transport protocol, replace *myQueue* with the name of the appropriate queue.

   You can make these changes without requiring any change to the web services binding file.

   If you are changing the port name and namespace without regenerating the WSBind file, the monitoring information might be wrong at runtime level 2.1 onwards.
3. Consider whether the automatically generated names in the WSDL document are appropriate for your purposes. You can rename these values:
   - 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 in the WSDL document.

   These values form part of the programmatic interface to which you code a client program. If the generated names are not sufficiently meaningful, maintenance of your application code might be more difficult over a long period of time. Use the DFHLS2WS **REQUEST-NAMESPACE** and **RESPONSE-NAMESPACE** parameters to change the `targetNamespace` of the XML schemas, and the **WSDL-NAMESPACE** parameter to change the `targetNamespace` of the WSDL document.

   If you change any of these values, you must use DFHWS2LS to regenerate the web services binding file. The language structures that are produced will not be the same as your existing language structures, 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 consider removing any fields that are not helpful to a web service client developer:
   - Fields that are used only for output values can be removed from the schema that maps the input data structures.
   - Filler fields.
   - Automatically generated annotations.

If you make any of these changes, you must 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 must write a wrapper program to map data from the new representation to the old one. This wrapper program needs to perform an **EXEC CICS LINK** command 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.

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. When you override the mapping rules, ensure 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.

Results

If you want to publish your customized WSDL document to an IBM WebSphere Service Registry and Repository (WSRR) server, you must publish it manually using the WSRR interface. You can find more information about WSRR at the following location: WebSphere Service Registry and Repository.

Example

For an example of a WSDL document, see An example of the generated WSDL document.

Sending a SOAP fault:

In a service provider, you can use the CICS API to send a SOAP fault to a web service requester. The fault can be issued by the service provider application or by a header processing program in the pipeline.

Before you begin

To use the API, the service provider application must use channels and containers. If the application uses COMMAREAs, write a wrapper program that does use channels and containers to create the SOAP fault message. You can use the API in a header processing program only if it is invoked directly from a CICS-supplied SOAP message handler.

About this task

You might want to issue a SOAP fault to the web service requester if your application logic cannot satisfy the request, for example, or if there is an underlying problem with the request message. Note that CICS does not consider issuing a SOAP fault as an error, so the normal message response pipeline processing takes place rather than any error processing. If you do want to roll back any transactions, you must use the application program.

Procedure

1. In your program, use the **EXEC CICS SOAPFAULT CREATE** command to send a SOAP fault, see SOAPFAULT CREATE.

2. Add the CLIENT or SERVER option on the command. This option indicates where the problem has occurred, either on the client side or on the server.
   - CLIENT indicates that the problem is with the request message that was received.
   - SERVER indicates that the problem occurs when the request message was processed by CICS. This problem might be in an application program, for example, it might be unable to satisfy the request, or it might be an underlying problem that occurs during the pipeline processing.
3. Add the FAULTSTRING option and its length in the FAULTSTRLEN option to provide a summary of why the fault has been issued by the service provider. The contents of this option are in XML. Any data supplied by the application must be in a format that is suitable for direct inclusion in an XML document. The application might have to specify some characters as XML entities. For example, if the < character is used anywhere other than the start of an XML tag, the application must change it to <. The following example shows an incorrect FAULTSTRING option:

   ```
   dcl msg_faultString char(*) constant('Error: Value A < Value B');
   ```

   The correct way to specify this FAULTSTRING option is as follows:

   ```
   dcl msg_faultString char(*) constant('Error: Value A &lt; Value B');
   ```

   Tip: To avoid using XML entities, you can wrapper the data in an XML CDATA construct. XML processors do not parse character data in this construct. Using this method, you could specify the following FAULTSTRING option:

   ```
   dcl msg_faultString char(*) constant('<![CDATA[Error: Value A < Value B]]>');
   ```
4. Code the DETAIL option and its length in the DETAILLENGTH option to provide the details of why the fault has been issued by the service provider. The contents of this option are in XML. The same guidance applies to the DETAIL option as to the FAULSTRING option.
5. Optional: If you are invoking the API from a header processing program, define the program in the pipeline configuration file. The header processing program is defined in either the <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> element.

Results

When your program issues this command, CICS creates the SOAP fault response message at the appropriate SOAP level. If your service provider application issues the command, it does not need to create a SOAP response and put it in the DFHRESPONSE container. The pipeline processes the SOAP fault through the message handlers and sends the response to the web service provider.

Example

The **SOAPFAULT CREATE** command has a number of options to provide you with flexibility to respond appropriately to a web service requester. Here is a simple example of a completed command that creates a SOAP fault that can be used for both SOAP 1.1 and SOAP 1.2:

```
EXEC CICS SOAPFAULT CREATE CLIENT
     DETAIL(msg_detail)
     DETAILLENGTH(length(msg_detail))
     FAULTSTRING(msg_faultString)
     FAULTSTRLEN(length(msg_faultString));
```

You can code *msg_detail* and *msg_faultString* with the following values:

```
dcl msg_detail char(*) constant('<ati:ExampleFault xmlns="http://www.example.org/faults"
xmlns:ati="http://www.example.org/faults">Detailed error message goes here.</ati:ExampleFault>');
dcl msg_faultString char(*) constant('Something went wrong');
```

Creating a web service requester using the web services assistant

You can create a service requester application from a web service description that complies with WSDL 1.1 or WSDL 2.0. The CICS web services assistant helps you to deploy your CICS applications in a service requester setting.

Before you begin

Your web services description must be in a file in z/OS UNIX or it must be published on an IBM WebSphere Services Registry and Repository (WSRR) server, and a requester mode pipeline must be installed in the CICS region.

You must allocate sufficient storage to the user ID so that the ID can run Java. You can use any supported version of Java. By default, DFHWS2LS uses the Java version specified in the **JAVADIR** parameter.

About this task

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.

Procedure

1. Use the DFHWS2LS batch program to generate a web service binding file and one or more language structures. Consider these options when creating a service requester application from a web service description:
   - Which mapping level do you want to use? The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to use the highest level of mapping available.
   - Which code page do you want to use when transforming data at run time? If you want to use a specific code page for your application that is different from the code page for the CICS region, use the **CCSID** parameter.
   - Do you want to support a subset of the operations that are declared in the web service description? If you have a very large web service description, and want your service requester application to support only a certain number of operations, use the **OPERATION** parameter to list the ones you want. Each operation must be separated with a space and is case sensitive.
   - Where is the WSDL document stored? If the WSDL document that you want to use as input to DFHWS2LS is stored on a WSRR server, you can retrieve it by running DFHWS2LS with certain parameters specified. Use the **WSRR-SERVER** parameter to specify the location of the WSRR server and use the **WSRR-NAME** parameter to specify the name of the WSDL document that you want to retrieve. For information about other parameters on DFHWS2LS that you might want to use to interact with WSRR, see "DFHWS2LS: WSDL to high-level language conversion" on page 66.
   - If you want to retrieve the WSDL document from a WSRR server, do you want to do so using a secure connection? You can use secure socket layer (SSL) encryption with the web services assistant to interoperate securely with

WSRR. For an example, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

Do not specify parameters such as **PROGRAM**, **URI**, **TRANSACTION**, and **USERID** when you use DFHWS2LS. These parameters apply only to a service provider application and, if specified, cause a provider mode web service binding file to be produced.In addition to the web service binding file, the program generates a language data structure.

2. Check the log file to see whether any problems occurred when DFHWS2LS generated the binding file and language structures. CICS might not support some elements or options in the web service description. If any warning or error messages are issued, read the advice that is provided and take appropriate action. You might need to rerun the batch program.
3. Copy the web service binding file to the pickup directory of the requester mode pipeline that you want to use for your web service application.
4. Ensure that the PIPELINE resource is configured for service requester applications. The value of the **MODE** parameter shows whether the pipeline supports requester or provider web service applications.
5. Ensure that the correct SOAP protocol is supported in the pipeline for your web service. The **SOAPLEVEL** parameter indicates which version is supported. In service requester mode, the binding of the web service must match the version of SOAP that is supported in the pipeline. You cannot install a web service with a SOAP 1.1 binding into a service requester pipeline that supports SOAP 1.2.
6. Ensure that the configured timeout for the pipeline is suitable for your service requester application. The timeout is displayed as the value of the RESPWAIT attribute on the PIPELINE resource. If no timeout is specified on the pipeline, the default for the transport is used.
   - The default timeout for HTTP is 10 seconds.
   - The default timeout for WebSphere MQ is 60 seconds.

   Each transaction in the CICS region has a dispatcher timeout. If this value is less than the default for either protocol, the timeout occurs with the dispatcher.
7. Optional: Copy the web service description to the same pickup directory as the web service binding file, so that you can turn on validation for the web service at run time.
8. Create the WEBSERVICE resource. This resource encapsulates the web service binding file in CICS and is used at run time.

   You can do this in the following ways:

   a. Using the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource.
   b. Defining the resource yourself. If you use the CICS Explorer to define a WEBSERVICE resource in a CICS bundle, you can choose to import a web service binding file and a WSDL document or WSDL archive file and include these in the bundle. You can then generate URIMAP definitions to support the web service and package these in a bundle. For more help with using the CICS Explorer to create and edit resources in CICS bundles, see Working with bundles in the CICS Explorer product documentation.
9. Write a wrapper program that you can substitute for your communications logic. Use the language data structure generated in step 1 to write your wrapper program. Use an **EXEC CICS INVOKE SERVICE** command in your wrapper program to communicate with the web service. The command includes these options:
   - The URI of the web service

- The operation for which the web service is being called

When you call the web service, you can specify a URIMAP resource that contains the information about the URI of the web service. You can specify this information directly on the INVOKE SERVICE command instead of using a URIMAP resource. However, using a URIMAP resource means that you do not need to recompile your applications if the URI of a service provider changes. With a URIMAP resource you can also choose to implement connection pooling, where CICS keeps the client connection open after use, so that it can be reused by the application for subsequent requests, or by another application that calls the same service. The PIPELINE SCAN command does not create URIMAP resources for a service requester, so you must define the URIMAP resource yourself following the instructions in Creating a URIMAP resource for CICS as a HTTP client in Developing applications.

Results

When you have successfully created the CICS resources, the creation of your service requester application is complete.

Checking the configuration of a PIPELINE resource

You can check the configuration of a PIPELINE with the following interfaces:

CICS Explorer

The CICS Explorer administration views

Use the Pipelines view.

CICSPlex® SM

The PIPELINE definitions view

CEMT

The INQUIRE PIPELINE command

The CICS SPI

The INQUIRE PIPELINE command

Creating a web service using tooling

Instead of using the web services assistant JCL, you can use Rational Developer for z Systems or write your own Java program to create the required files in CICS.

Procedure

1. You have two choices:
   - Use the Rational Developer for z Systems tool to create a web service binding file and the web service description or language structures. For more information about this tool, see Rational Developer for System z.
   - Write your own Java program, using the provided API, to invoke the web services assistant. This API is described in the Web services assistant: Class Reference Javadoc. It includes comments that explain the classes, and sample code is provided to give an example of how you might invoke the web services assistant. The Javadoc also contains a complete list of the JAR files that are required and their location in z/OS UNIX.

You can run your Java program on the z/OS, Windows, or Linux platform. If you run the program on Windows or Linux, transfer the generated web services binding file to a suitable pickup directory in binary mode using FTP or an equivalent process.

2. Optional: If you are generating a web service description from a language structure, review the file and perform any necessary customization. For more information, see "Customizing generated web service description documents" on page 135.
3. Deploy the generated web service binding file into a suitable pipeline pickup directory.
4. Optional: Copy the web service description into the pickup directory of the pipeline, so that you can perform validation of the web service to check that it is working as expected.
5. If you started with a web service description, write a service provider or requester application program to interface with the generated language structures.
6. Run a **PIPELINE SCAN** command to automatically create the required CICS resources.

Creating your own XML-aware web service applications

If you decide not to use the CICS-supplied data mappings, you can write your own XML-aware data applications in two ways instead. You can either use the **XML-ONLY** parameter on DFHWS2LS or you can write your own application without using any of the tooling. Using the **XML-ONLY** parameter is the most straightforward way to configure the CICS pipeline process to pass the XML data to the application to be handled.

About this task

Writing your own XML-aware applications involves writing code to both parse and generate XML documents. One way to write your own XML-aware application uses the `XML PARSE` and `XML GENERATE` statements in COBOL. Another way to write your own XML-aware applications uses other IBM tools; for example, you can use the Rational Developer for z Systems for tool to generate COBOL XML converter programs that can be invoked from your applications.

Creating an XML-aware service provider application:

Your XML-aware service provider application must work with the containers that are passed to it and handle the data conversion between the XML and the program language.

About this task

The following steps guide you through the creation of your XML-aware application, including the decision about the use of any of the CICS tooling.

Procedure

1. Decide if you want to generate a web service binding file for your XML-aware application using DFHWS2LS. The advantage of generating a web service binding file is that you can use CICS services, such as validation, to test your web service and CICS monitoring using global user exits.
   - If you want to generate a web service binding file, run DFHWS2LS specifying the **XML-ONLY** parameter and a **MINIMUM-RUNTIME-LEVEL** of 2.1 or

higher. The web service binding file enables the application program to work directly with the contents of the DFHWS-BODY container. In all other respects, the generated binding file shares the same deployment characteristics and the same runtime behavior as a file generated without the **XML-ONLY** parameter, including parsing of the XML during SOAP message handling. To prevent this parsing, you must not specify SOAP message handlers in your pipeline configuration file.

- If you do not want to use a web service binding file, configure your service provider pipeline so that the web service request reaches your XML-aware application. You can either configure the terminal handler in the pipeline configuration file to use your XML-aware application program or create a message handler that dynamically switches to your application depending on the URI that is received in the pipeline.

2. Write your application to handle the web service request that is held in the following containers:

   DFHWS-BODY
   The contents of the SOAP body for an inbound SOAP request when the pipeline includes a CICS-provided SOAP message handler.

   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.

   When you code API commands to work with the containers, do not 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.

3. Optional: Your application can also use additional containers that are available to message handlers in the pipeline, as well as any other containers that the message handlers create as part of their processing. For a complete list of containers, see "Containers used in the pipeline" on page 290.

4. When your application has processed the request, construct a web service response using the following containers:

   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 instead of using the CICS-provided SOAP message handler.

   If you supply a SOAP body in container DFHWS-BODY, DFHRESPONSE is ignored.

   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 constructs the full SOAP message containing the body.

Your program must create this container, even if the request and response are identical. If you do not, CICS issues an internal server error.

You can also use any of the other containers to pass information that your pipeline needs for processing the outbound response.

If 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, because the message handler checks only whether the container is present or not.

5. Create a URIMAP resource. If you are using the **XML-ONLY** parameter and you have specified a value for the **URI** parameter of DFHWS2LS, the URIMAP is created automatically for you during the PIPELINE SCAN process.

Creating an XML-aware service requester application:

Your XML-aware web service requester application handles the data conversion between the XML and the programming language and populates the control containers in the pipeline.

Before you begin

You can write your own XML-aware service requester application using the **XML-ONLY** parameter on DFHWS2LS or you can write it without using any of the tooling. The most straightforward way to write your own XML-aware service requester application is by using the **XML-ONLY** parameter on DFHWS2LS; the **XML-ONLY** parameter is available at runtime level 2.1 and later.

About this task

Using the **XML-ONLY** parameter results in the generation of a WSBind file that instructs CICS that the application will work directly with the contents of the DFHWS-BODY container. The generated WSBind file must be installed into a requester mode PIPELINE to create a requester mode WEBSERVICE resource. The application must generate XML for the body of the web service request and store it in the DFHWS-BODY container. It must then call the **EXEC CICS INVOKE SERVICE** command. The outbound message is sent to the web services provider. The body of the response message is also in the DFHWS-BODY container after the call completes.

The XML of the response messages is parsed during SOAP message handling. To prevent this parsing, you must not specify SOAP message handlers in your pipeline configuration file.

XML-aware requester applications can receive SOAP Fault messages back from the remote provider mode application. In this case, the requester application is responsible for interpreting the SOAP Fault and distinguishing it from a regular response message. If the **INVOKE SERVICE** command is used with an **XML-ONLY** WEBSERVICE, CICS does not set the response code which is normally used to indicate that a SOAP Fault was received.

If you are writing your own XML-aware service requester application without using the **XML-ONLY** option, complete the following steps:

Procedure

1. Create a channel and populate it with containers. The control containers must all be populated in CHAR mode. Provide the following information in each container:

   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 constructs 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. The pipeline must not include a CICS-provided SOAP message handler to avoid duplicate SOAP headers being sent in the outbound message.

   If you supply a SOAP body in container DFHWS-BODY, DFHREQUEST must 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.

   Tip: If you add container DFHWS-NOABEND to the channel, when DFHPIRT is called any abends will not be issued from within DFHPIRT. This is useful if you are running a C/C++ program because you can handle errors via the DFHERROR container.

2. Link to program DFHPIRT. Use this command:

   ```
   EXEC CICS LINK PROGRAM(DFHPIRT) CHANNEL(userchannel)
   ```

   where *userchannel* is the channel that holds your containers. The outbound message is processed by the message handlers and header processing programs in the pipeline and sent to the web service provider.

3. Retrieve the containers that contain the web service response from the same channel. The response from the web service provider might be a successful response or a SOAP fault. The web service requester application must be able to handle both types of response from the service provider. The complete response is contained in the following containers:

   DFHRESPONSE
   The complete response, including the envelope for a SOAP response, received from the web service provider.

DFHWS-BODY
When the pipeline includes a CICS-provided SOAP message handler, the contents of the SOAP body.

DFHERROR
Error information from the pipeline.

Note: In some error cases DFHWS-BODY might not be updated. You must check DFHRESPONSE for a SOAP fault.

Using Java with web services

You can use Java to create web service applications. Different techniques are used to create these applications compared with the techniques used with other programming languages.

For most non-Java programming languages, you use the web services assistants to enable applications. Using the web services assistant means CICS will convert the data from the web service into a form suitable for the application and place it into a container or COMMAREA. You can use the web services assistant with Java applications, however, the following tasks provide more suitable methods for creating Java web services for Java applications.

Deploying a Java provider-mode web service in an Axis2 JVM server:

You can deploy an Axis2 application as a provider mode web service in CICS. These applications are typically generated using JAX-WS and can be hosted in a Java enabled pipeline.

You might want to deploy Java applications using this method for one of the following reasons:

- You have existing investment using Axis2 Handler interfaces.
- You want to use a CICS pipeline configuration.

Note: Axis2-style applications do not use the WEBSERVICE resources. They interact with CICS using the Axis2 programming model and therefore cannot use the some of the CICS web services support. The following services are not fully supported for Axis2-style applications:

- SOAPFAULT CREATE
- WSACONTEXT GET
- "DFHWS-OPERATION container" on page 302
- "DFHWS-MEP container" on page 302
- "DFHWS-USERID container" on page 308
- "DFHWS-TRANID container" on page 303
- Web services security

Before you begin

You must have a Java application that is suitable for deployment in Axis2, for example a POJO application using JAX-WS. For this task, the following POJO application is used as an example:

```
/**
 * Simple example
 */
@javax.jws.WebService(targetNamespace = "com.ibm.cics.example", name = "pojoExample")
public class TestAxis2
```

```
{
    public String getMessage(String input)
    {
        return "CICS got this: '" + input + "'";
    }
}
```

This application specifies the XML namespace that is used to generate the WSDL, and a name to associate with the web service.

The Java code for this application must be compiled, and the JAX-WS generator run, to package the application into a jar file called `TestAxis2.jar`. You can do this by issuing the following code:

```
javac TestAxis2.java
wsgen -cp . TestAxis2 -wsdl
jar -cvf TestAxis2.jar *
```

The JAX-WS generator also creates a WSDL document and the bindings used by Axis2.

About this task

To deploy an Axis2 web service you must create the pipeline infrastructure for your web services. When you have created the pipeline, you can create your web services. You can reuse the created pipeline for as many web services as you need. The following steps describe how to create the pipeline and web services.

Note: No WEBSERVICE resource is created or installed as part of this task.

Procedure

1. Create the pipeline infrastructure.
   a. Create a web service infrastructure for a Java pipeline. For more information, see "Creating the CICS infrastructure for a SOAP service provider" on page 217.
   b. Create an Axis2 repository. To do this, create a copy of the supplied repository located in `$CICS_HOME/lib/pipeline/repository`.
   c. Add the `<repository>` element to your pipeline configuration file. This element must specify the name of the Axis2 repository that you created.
   d. Create and enable a PIPELINE resource.
2. For each web service associated with the pipeline, repeat the following steps to create the web service.
   a. Deploy the Axis2 application to the Axis2 repository. For example, the jar file created in the example must be deployed to a directory called `servicejars` in the repository directory. You must create this directory if it does not exist.
   b. Define and install a URIMAP resource for the web service. The URIMAP resource must specify the URI and PIPELINE resource associated with the web service. The URI must follow the Axis2 naming conventions for URIs. The default Axis2 naming convention is: `/`*`name_of_service`*`Service.`*`name_of_port`*`Port/`*`suffix`*, where *name_of_service* is the name of the web service in the WSDL, *name_of_port* is the name of the port in the WSDL and *suffix* is an optional suffix that you can define. For the preceding example, the following URIMAP resource could be used:

```
Urimap          : EXAMPLE
Group           : EXAMPLE
STatus          : Enabled
USAge           : Pipeline
SCheme          : HTTP
POrt            : No
HOST            : *
PAth            : /TestAxis2Service.pojoExamplePort/example/TestAxis2
TRansaction     : CPIH
PIpeline        : EXAMPLE
```

This example assumes that the PIPELINE resource used is called EXAMPLE.

What to do next

Test that your web services run correctly.

Creating a Java web service that generates and parses XML:

You can create Java applications that parse and generate XML themselves. These applications are consistent with XML-aware applications written in other programming languages, but they benefit from using standard Java technologies for processing the XML.

Procedure

1. Create an XML-ONLY WEBSERVICE resource. For more information, see "Creating an XML-aware service requester application" on page 144 or "Creating an XML-aware service provider application" on page 142.
2. Write a Java web service that can parse and generate XML for the body of the SOAP message. You can use various tools, such as the Java 6 Java Architecture for XML Binding (JAXB) library, to help you create a Java web service with these capabilities.
3. Optional: If you are using a provider pipeline and you want to add the capability for a SOAP Fault message to be returned to the requester, use the JCICS SoapFault class to issue the **EXEC CICS SOAPFAULT CREATE** command.
4. Optional: If you are using a requester pipeline, use the JCICS Service class to issue the **EXEC CICS INVOKE SERVICE** command.

Creating a Java web service that has a COBOL interface:

You can create Java applications that interact with CICS using the same techniques used in other programming languages. To create these applications, you must write or generate Java code to create structured COMMERA- or container-style data.

Procedure

1. Use DFHWS2LS to create COBOL language structures for the web service.
2. Write a Java web service that generates and parses COBOL language structures. For more information about tools that allow Java programs to access existing CICS application data and links to examples of how to create a Java web service that can generate and parse COBOL language structures, see Interacting with structured data from Java in Developing applications.
3. Optional: If you are using a provider pipeline and you want to add the capability for a SOAP Fault message to be returned to the requester, use the JCICS SoapFault class to issue the **EXEC CICS SOAPFAULT CREATE** command.

4. Optional: If you are using a requester pipeline, use the JCICS Service class to interface with the CICS SERVICE API and issue the **EXEC CICS INVOKE SERVICE** command.

Deploying a requester-mode JAX-WS web service:

You can deploy a JAX-WS application as a requester mode web service in CICS. However, these applications do not use the **EXEC CICS INVOKE** command, instead they interact with the remote web services using JAX-WS.

Before you begin

You must have a JVM server configured to support OSGi. For more information, see Setting up a JVM server in Configuring.

About this task

The advantage of deploying a JAX-WS application as a requester mode web service is that you create a platform-independent web service requester application, which uses the zEnterprise Application Assist Processor (zAAP). Using zAAP can reduce the cost of transactions; for more information, see the IBM Redbooks publication: zSeries Application Assist Processor (zAAP) Implementation.

Procedure

1. Create a web service requester application in Java and use an appropriate API, such as the Java API for XML web services (JAX-WS), to call the remote web service.
2. Optional: If you use JAX-WS to start a remote web service, you must also use JAX-WS to generate the SOAP messages, handle the network communication, and process the SOAP response.
3. Deploy your Java application and install it in the JVM server.

What to do next

Test that your web services start correctly.

Deploying a Java provider-mode web service in a Liberty JVM server:

You can deploy a web application as a provider mode web service in a Liberty JVM server. These applications are created using Java standards JAX-WS and JAXB.

About this task

CICS TS V5.3 includes the latest WebSphere Application Server Liberty Profile (WLP) V8.5.5 that provides features for the Java API for XML Web Services (JAX-WS) and the Java Architecture for XML Binding (JAXB). Together these technologies enable you to write SOAP web services in Java as part of a CICS application. The following article will show you how to set up Eclipse, test a sample web services project, deploy the sample into CICS, modify the sample to use JCICS and test it with the Web Service Explorer. For details see DeveloperWorks, JAX-WS and JAXB support in CICS TS V5.2 open beta Liberty profile.

You might want to deploy Java applications using this method for one of the following reasons:

- You want to create web services in Java.
- You have complicated WSDL documents that would be difficult to handle using the CICS web services assistants.
- You want to offload the handling of the web service application to the zEnterprise Application Assist Processor (zAAP).

Note: Web applications deployed to a Liberty JVM server do not use the WEBSERVICE or TCPIPSERVICE resources. They interact with web requests using the Liberty HTTP listener and therefore cannot use the facilities of the CICS web services support.

Validating SOAP messages

When you use the CICS web services you can specify that the SOAP messages are to be validated to ensure that they conform to the schema that is contained in the web service description. You can validate both provider and requester mode applications.

Before you begin

During development and testing of your web service deployment, full validation assists in detecting problems in the message exchange between a service requester and a service provider. However, complete validation of the SOAP messages carries a substantial overhead, and it is inadvisable to validate messages in a fully tested production application.

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

About this task

The SOAP message is validated 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 and is validated again against the transformation requirements of CICS. You can use the WSDL file specified in the **WSDLFILE** attribute of the WEBSERVICE resource or a WSDL file contained in the `.zip` file specified in the **ARCHIVEFILE** attribute of the WEBSERVICE resource. If both attributes are specified, the WSDL file in the archive file specified in the **ARCHIVEFILE** attribute is used.

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. Therefore a SOAP message might be successfully validated but then fail in the runtime environment and vice versa.

Procedure

1. Optional: Set up a JVM server in the CICS region. You can run SOAP validation in an OSGi framework or Axis2, but not in a Liberty profile. CICS provides samples to quickly set up a JVM server that uses an OSGi framework. If you are using a Java pipeline, there is no need to define an extra JVM server.
   - When you have a JVM server already defined that you want to use for validation, modify the DFHPIVAL program definition in group DFHPIVAL to reference the name of the JVMSERVER resource. The DFHPIVAL definition is not locked and can be edited. By default, the definition references DFHJVMS.

     or

- When you want to use the CICS supplied sample you must install the sample JVM server DFHJVMS in group DFH$OSGI. For more information, see Setting up a JVM server in Configuring.

2. Ensure that you have a web service description associated with your WEBSERVICE resource. This association is created for WEBSERVICE resources that are automatically created when a WSDL file or a `.zip` file that contains one or more WSDL files is present in the pickup directory of the pipeline during a pipeline scan.

   For WEBSERVICE definitions that are created with RDO, the web service description is specified with the WSDLFILE attribute.
3. Turn web service validation on by specifying **`VALIDATION=YES`** attribute of the WEBSERVICE resource. You can specify whether validation is required when you define the resource, and you can change this setting after the resource is installed.

Results

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

What to do next

Turn validation off when you no longer need it.

Changing the validation status of a web service

You can change the validation status of a web service with the following interfaces:

CICS Explorer

The CICS Explorer administration views

Use the **`Validation Status`** attribute in the Web Services view.

CICSPlex SM

The WEBSERVICE definitions view

CEMT

The SET WEBSERVICE command

The CICS SPI

The SET WEBSERVICE command

Creating a JSON web service

You can expose existing CICS applications as JSON web services and create new CICS applications to act as JSON web service providers.

Before you begin

Before you begin to create a JSON web service, you must configure your CICS system to support JSON web services. For more information, see “Creating the

CICS infrastructure for a JSON service provider" on page 221.

About this task

The CICS JSON assistant is a supplied utility that helps you to create the necessary artifacts for a new JSON web service provider application, or to enable an existing application as a JSON web service provider.

The CICS JSON assistant can create a JSON schema from a high-level language structure or a high-level language structure from an existing JSON schema; it supports COBOL, C/C++, and PL/I. It also generates information that is used to enable automatic runtime conversion of the JSON messages to containers and COMMAREAs, and vice versa. This information is used by the CICS JSON web services support during pipeline processing.

Create your JSON web service, as described in the following procedure, and validate that it works correctly:

Procedure

1. Create a JSON web service. Use the JSON assistant to create the JSON schema or language structures and deploy them into CICS. Use the **PIPELINE SCAN** command to automatically create the required CICS resources.
2. Start the JSON web service to test that it works as you intended.

What to do next

These steps are explained in more detail in the following topics.

The CICS JSON assistant

The CICS JSON assistant is a set of batch utilities that creates a mapping between JSON schema and language structures. This mapping is used by CICS at runtime to do the transformation between JSON and application data. 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 JSON 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 JSON message and the application program's data structure.

The assistant can create a JSON schema from a high-level language structure or a high-level language structure from an existing JSON schema, and supports COBOL, C/C++, and PL/I. It also generates information used to enable automatic runtime conversion of the JSON messages to containers and COMMAREAs, and vice versa.

The CICS JSON assistant comprises two utility programs:

DFHLS2JS
: Generates a web service binding file from a language structure. This utility also generates a JSON schema.

DFHJS2LS
: Generates a web service binding file from a JSON schema. 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.

For more information about the JSON assistant utility programs and data mappings, see the following topics.

DFHLS2JS: High-level language to JSON schema conversion for request-response services:

The DFHLS2JS procedure generates a JSON schema file from a high-level language data structure. You can use DFHLS2JS when you expose a CICS application program as a service provider.

The job control statements for DFHLS2JS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

The DFHLS2JS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

The relevant usage mode for the DFHLS2JS or DFHJS2LS procedure depends on your requirements:

- DFHLS2JS: High-level language to JSON schema conversion for linkable interface
- DFHJS2LS: JSON schema to high-level language conversion for linkable interface
- DFHLS2JS: High-level language to JSON schema conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for RESTful services

Job control statements for DFHLS2JS

JOB Starts the job.

EXEC Specifies the procedure name (DFHLS2JS).

INPUT.SYSUT1 DD
Specifies the input. The input parameters are typically 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 DFHLS2JS:

JAVADIR=*path*
Specifies the name of the Java directory that is used by DFHLS2JS. The value of this parameter is appended to /usr/lpp/ to produce a complete path name of /usr/lpp/*path*.

Typically, you do not 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 a prefix that extends the z/OS UNIX directory path that is used on other parameters, or '' (empty string) if no prefix is used.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHPREF** parameter.

SERVICE=*value*
Use this parameter only when directed to do so by IBM Support.

TMPDIR=*tmpdir*
Specifies the location of a directory in z/OS UNIX that DFHLS2JS uses as a temporary workspace. The user ID used to run the job must have read and write permission to this directory.

The default value is `/tmp`.

TMPFILE=*tmpprefix*
Specifies a prefix that DFHLS2JS uses to construct the names of the temporary workspace files.

The default value is `LS2JS`.

USSDIR=*path*
Specifies the name of the CICS TS directory in the UNIX System Services file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*. This must be specified as `'.'` (period) if the default is used.

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

The temporary workspace

DFHLS2JS creates the following three temporary files at run time:

tmpdir`/`*tmpprefix*`.in`

tmpdir`/`*tmpprefix*`.out`

tmpdir`/`*tmpprefix*`.err`

Where:

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

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

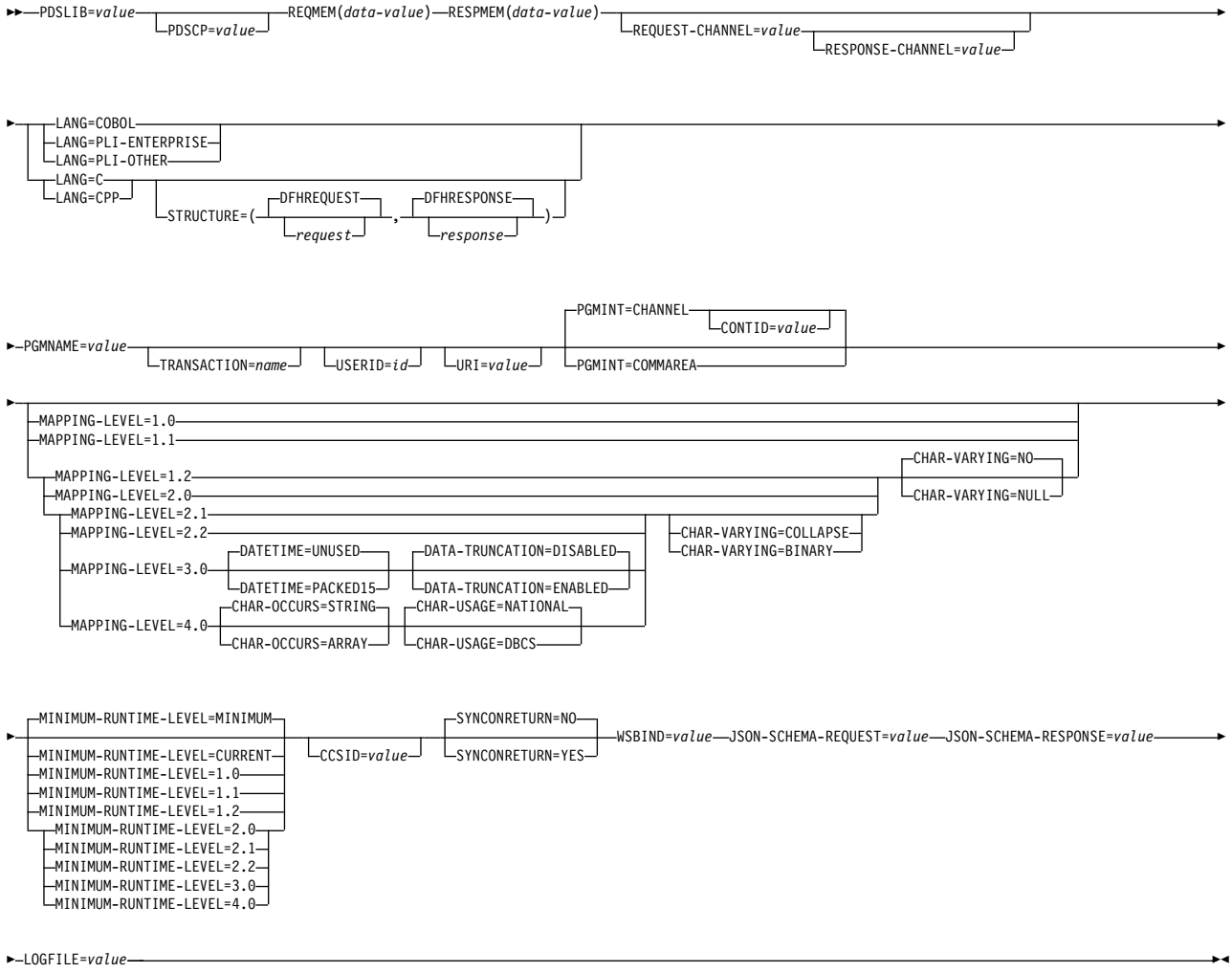
The default names for the files, when **TMPDIR** and **TMPFILE** are not specified, are as follows:

```
/tmp/LS2JS.in
/tmp/LS2JS.out
/tmp/LS2JS.err
```

Important: DFHLS2JS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHLS2JS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that 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 DFHLS2JS



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 - 80 must 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.

- A comma in the last character position of the line is an optional line separator, and 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 Unicode Services User's Guide and Reference. If you do not specify this parameter, the application data structure is encoded by using the CCSID specified in the system initialization parameter.

CHAR-VARYING={**NO**|**NULL**|**COLLAPSE**|**BINARY**}
Specifies how character fields in the language structure are mapped when the mapping level is 1.2 or higher. 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. You can select these options:

NO Character fields are mapped to a JSON string and are processed as fixed-length fields. The maximum length of the data is equal to the length of the field. NO is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping levels 2.0 and earlier.

This value does not apply to Enterprise and Other PL/I language structures.

NULL Character fields are mapped to a JSON string and are processed as null-terminated strings. CICS adds a terminating null character when transforming from a JSON message. The maximum length of the character string is calculated as one character less than the length indicated in the language structure. NULL is the default value for the **CHAR-VARYING** parameter for C/C++.

This value does not apply to Enterprise and Other PL/I language structures.

COLLAPSE
Character fields are mapped to a JSON string. Trailing white space in the field is not included in the JSON message. The inbound JSON message is parsed to remove all leading, trailing, and embedded white space. COLLAPSE is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping level 2.1 onwards.

BINARY
Character fields are mapped to a JSON string containing base64 encoded data and are processed as fixed-length fields. The BINARY value on the **CHAR-VARYING** parameter is available only at mapping levels 2.1 and onwards.

CHAR-OCCURS={**STRING**|**ARRAY**}
Specifies how character arrays in the language structure are mapped when the mapping level is 4.0 or higher. For example, `PIC X OCCURS 20`. This parameter is only for use by COBOL language.

ARRAY
Character arrays are mapped to a JSON array. This means that every character is mapped as an individual JSON element. This is also the behaviour at mapping levels 3.0 and earlier.

STRING
Character arrays are mapped to an JSON string. This means that the entire COBOL array is mapped as a single JSON element.

CHAR-USAGE={**NATIONAL**|**DBCS**}

In COBOL, the national data type, `PIC N`, can be used for UTF-16 or DBCS data. This setting is controlled by the `NSYMBOL` compiler option. You must set the **CHAR-USAGE** parameter on the assistant to the same value as the `NSYMBOL` compiler option to ensure that the data is handled appropriately. This is typically set to `CHAR-USAGE=NATIONAL` when you use UTF-16.

DBCS Data from `PIC (n)` fields is treated as UTF-16 encoded data.

NATIONAL
Data from `PIC (n)` fields is treated as DBCS encoded data.

CONTID=*value*
In a service provider, specifies the name of the container that holds the top-level data structure that is used to represent a JSON message.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed-length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**UNUSED**|**PACKED15**}
Specifies if potential ABSTIME fields in the high-level language structure are mapped as timestamps:

PACKED15
Packed decimal fields of length 15 (8 bytes) are treated as CICS ABSTIME fields, and mapped as timestamps.

UNUSED
Packed decimal fields of length 15 (8 bytes) are not treated as timestamps.

You can set this parameter at a mapping level of 3.0.

JSON-SCHEMA-REQUEST=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the request JSON schema is stored.

JSON-SCHEMA-RESPONSE=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the response JSON schema is stored.

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 z/OS UNIX name of the file into which DFHLS2JS writes its activity log and trace information. DFHLS2JS creates the file, but not the directory structure, if it does not exist.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHLS2JS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHLS2JS uses when you generate the web service binding file and JSON schema. You can select these options:

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 Use this mapping level to generate JSON schema using the full set of options available.

4.0 Use this mapping level with a CICS TS 5.2 region. At this mapping level you can use COBOL `OCCURS DEPENDING ON` fields and the **CHAR-OCCURS** parameter.

For more information about mapping levels, see Mapping levels for the CICS JSON assistants.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}
Specifies the minimum CICS runtime environment into which the web service

binding file can be deployed. If you select a level that does not match the other parameters that you specified, you receive an error message. You can select these options:

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

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 The generated web service binding file deploys into a CICS TS 4.1 region or later.

Note: JSON support is only available from CICS TS 4.2 onwards.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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 that are used for the request and response are specified in the **REQMEM** and **RESPMEM** parameters.

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

PDSCP=*value*
Specifies the code page that is used in the partitioned data set members that are 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, the z/OS UNIX System Services code page is used. For example, you might 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.

- In mapping levels earlier than 3.0, the channel can contain only one container, which is used for both input and output. Use the **CONTID** parameter to specify the name of the container. The default name is DFHWS-DATA.
- At mapping level 3.0, the channel can contain multiple containers. Use the **REQUEST-CHANNEL** and **RESPONSE-CHANNEL** parameters. Do not specify **PDSLIB**, **REQMEM**, or **RESPMEM**.

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

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of the CICS PROGRAM resource for the target application program that is exposed as a web service. The CICS web service support links to this program.

REQMEM=*value*
Specifies the name of the partitioned data set member that 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.

REQUEST-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when it receives a JSON message from a web service requester. The channel description is an XML document that must conform to the CICS supplied channel schema. For more information, see "Creating a channel description document" on page 187.

You can use this parameter at mapping level 3.0 only.

RESPMEM=*value*
Specifies the name of the partitioned data set member that 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.

RESPONSE-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when it sends a JSON response message to a web service requester. The channel description is an XML document that must conform to the CICS supplied channel schema. For more information, see "Creating a channel description document" on page 187.

You can use this parameter at mapping level 3.0 only.

STRUCTURE=(*request*,*response*)

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

request

Specifies the name of the high-level structure that contains 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 that contains 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 sync point.

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

YES The remote web service can issue a sync point. 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 one to four 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ # _ < >

URI=*value*

This parameter specifies the relative or absolute URI that a client uses to access the web service. CICS uses the value that is specified when it generates a URIMAP resource from the web service binding file that is created by DFHLS2JS. 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 one to eight 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 by using the **PIPELINE** scan command.

Acceptable characters: A-Z a-z 0-9 $ @ #

WSBIND=*value*
The fully qualified z/OS UNIX name of the web service binding file. DFHLS2JS creates the file, but not the directory structure, if it does not exist. The file extension is `.wsbind`.

Other information

- The user ID that DFHLS2JS uses to run must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX file structure and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **JSON Schema** parameters.
- The user ID must have a sufficiently large storage allocation to run Java.
- The JCL has a maximum parameter length of 100 characters. This parameter length can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

Example

```
//LS2JS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHLS2JS,
// TMPFILE=&amp;QT.&SYSUID.&QT,
//INPUT.SYSUT1 DD *
PDSLIB=CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
JSON-SCHEMA-REQUEST=/u/exampleapp/json/example_request.json
JSON-SCHEMA-RESPONSE=/u/exampleapp/json/example_response.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=COLLAPSE
PGMNAME=DFH0XCMN
URI=http://myserver.example.org:8080/exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

DFHJS2LS: JSON schema to high-level language conversion for request-response services:

The DFHJS2LS procedure generates a high-level language data structure and a web service binding file from a JSON schema. You can use DFHJS2LS when you prepare to create a CICS application program as a service provider.

The job control statements for DFHJS2LS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

The DFHJS2LS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

The relevant usage mode for the DFHLS2JS or DFHJS2LS procedure depends on your requirements:

- DFHLS2JS: High-level language to JSON schema conversion for linkable interface
- DFHJS2LS: JSON schema to high-level language conversion for linkable interface
- DFHLS2JS: High-level language to JSON schema conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for RESTful services

Job control statements for DFHJS2LS

JOB Starts the job.

EXEC Specifies the procedure name (DFHJS2LS).

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 DFHJS2LS:

JAVADIR=*path*
Specifies the name of the Java directory that is used by DFHJS2LS. The value of this parameter is appended to `/usr/lpp/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 a prefix that extends the z/OS UNIX directory path that is used on other parameters, or `''` (empty string) if no prefix is used.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHPREF** parameter.

SERVICE=*value*
Use this parameter only when directed to do so by IBM Support.

TMPDIR=*tmpdir*
Specifies the location of a directory in z/OS UNIX that DFHJS2LS uses as a temporary workspace. 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 DFHJS2LS uses to construct the names of the temporary workspace files.

The default value is `JS2LS`.

USSDIR=*path*
Specifies the name of the CICS TS directory in the UNIX System Services file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to

produce a complete path name of /usr/lpp/cicsts/*path*. This must be specified as '.' (period) if the default is used.

Typically, you do not 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

DFHJS2LS creates the following three temporary files at run time:

tmpdir/*tmpprefix*.in

tmpdir/*tmpprefix*.out

tmpdir/*tmpprefix*.err

where:

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

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

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

/tmp/JS2LS.in

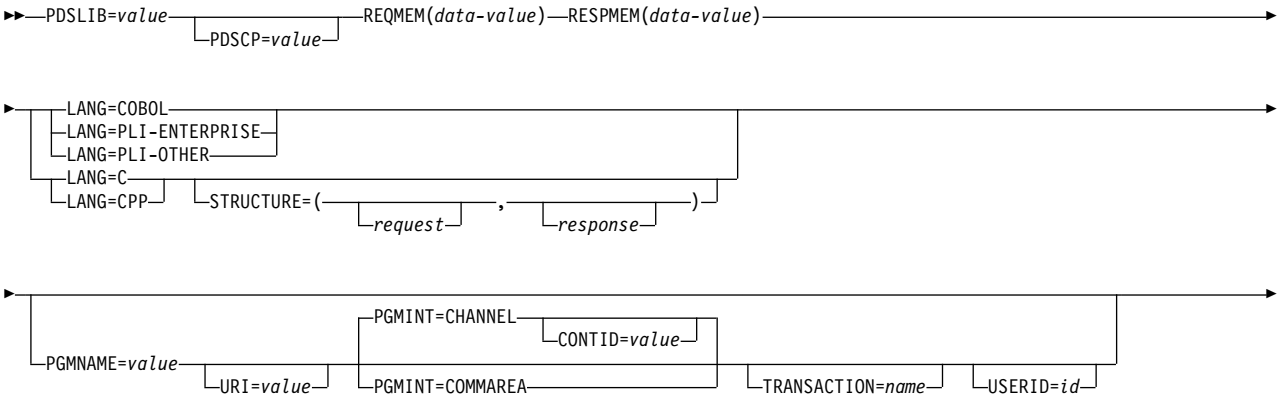
/tmp/JS2LS.out

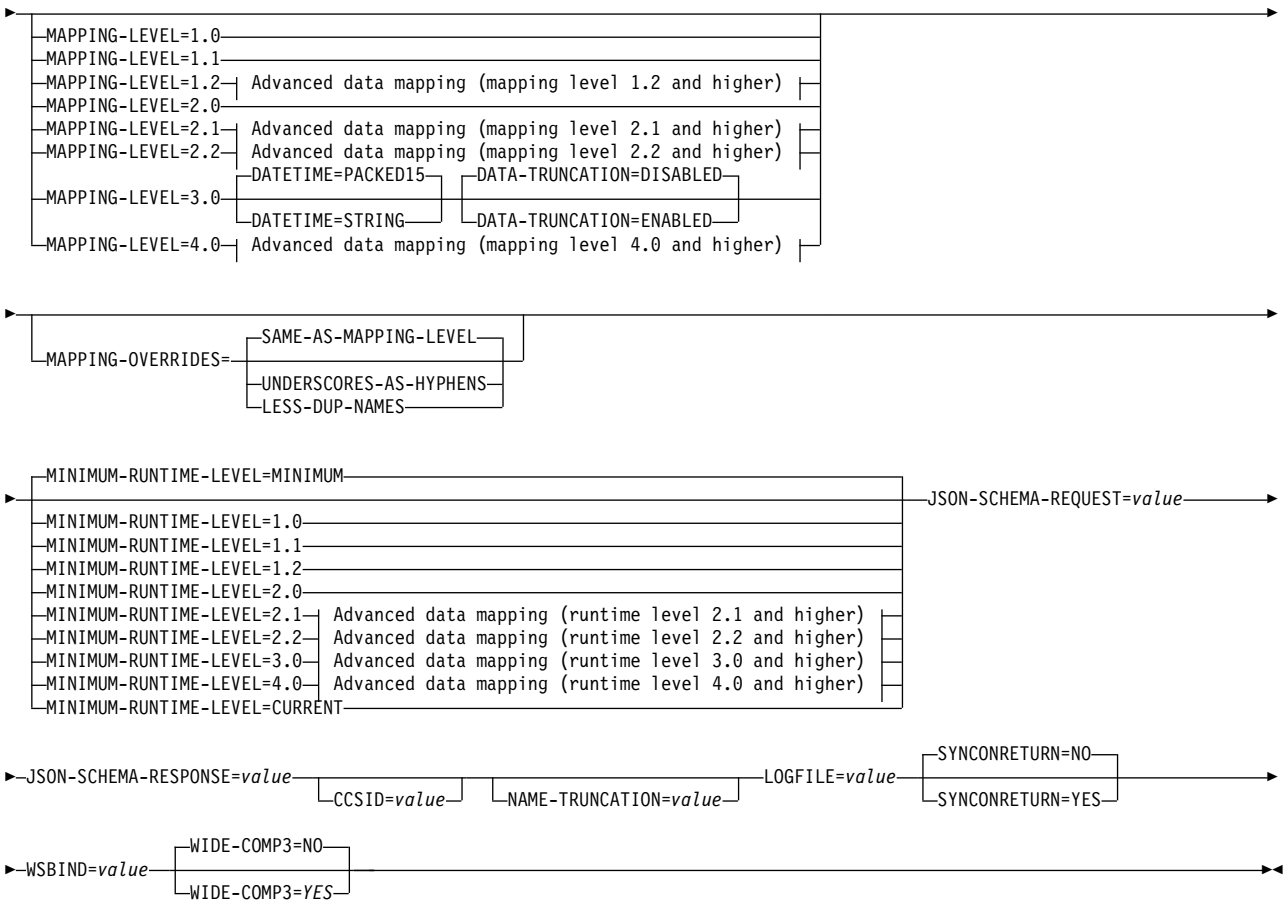
/tmp/JS2LS.err

Important: DFHJS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHJS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

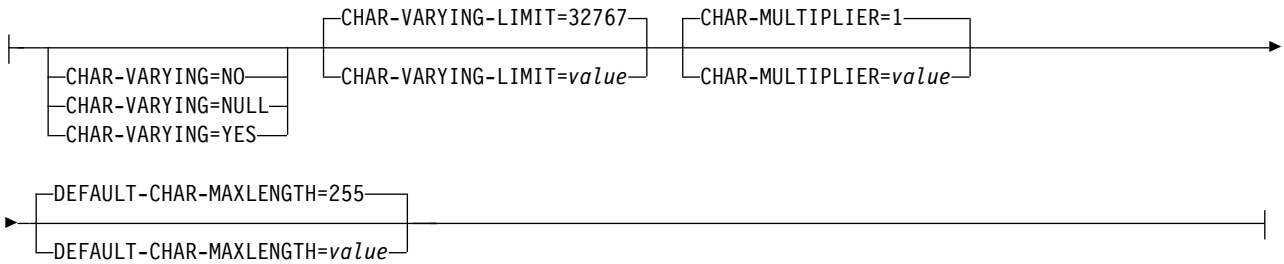
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of the job.

Input parameters for DFHJS2LS





Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):

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 - 80 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 Unicode Services User's Guide and Reference. If you do not specify this parameter, the application data structure is encoded by using the CCSID specified in the system initialization parameter.

CHAR-MULTIPLIER={**1**|*value*}
: Specifies the number of bytes to allow for each character when the mapping level is 1.2 or later. The *value* of this parameter can be a positive integer in the range of 1 - 2,147,483,647. 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 might 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.

  When you set **CCSID=1200** (indicating UTF-16), the only valid values for **CHAR-MULTIPLIER** are 2 or 4. When you use UTF-16, the default value is 2. Use **CHAR-MULTIPLIER=2** when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use **CHAR-MULTIPLIER=4** when you expect application data to contain characters that require 2 UTF-16 encoding units.

  Note: Setting **CHAR-MULTIPLIER** to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger **CHAR-MULTIPLIER** value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.

CHAR-VARYING={**NO**|**NULL**|**YES**}
: Specifies how variable-length character data is mapped when the mapping level is 1.2 or higher. Variable-length binary data types are always mapped to either a container or a varying structure. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

  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 two related elements: data-length and the data.

CHAR-VARYING-LIMIT={**32767**|*value*}
Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure when the mapping level is 1.2 or higher. If the character or binary 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 32,767 bytes.

CONTID=*value*
In a service provider, specifies the name of the container that holds the top-level data structure that is used to represent a JSON message.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed-length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**PACKED15**|**STRING**}
Specifies how JSON date-time elements are mapped to the language structure.

PACKED15
The default is that any JSON date-time element is processed as a timestamp and is mapped to CICS ABSTIME format.

STRING
The JSON date-time element is processed as text.

DEFAULT-CHAR-MAXLENGTH={**255**|*value*}
Specifies the default array length of character data in characters for mappings where no length is implied in the web service description document, when the mapping level is 1.2 or higher. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647.

INLINE-MAXOCCURS-LIMIT={**1**|*value*}
Specifies whether inline variable repeating content is used based on the `maxItems` JSON schema keyword. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.

The **INLINE-MAXOCCURS-LIMIT** parameter is available only at mapping level 2.1 onwards. The value of **INLINE-MAXOCCURS-LIMIT** can be a positive integer in the range of 0 - 32,767. A value of `0` indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the *value* of the `maxOccurs` attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise inline mapping is used.

When deciding whether you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.

JSON-SCHEMA-REQUEST=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the request JSON schema is stored.

JSON-SCHEMA-RESPONSE=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the response JSON schema is stored.

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 z/OS UNIX name of the file into which DFHJS2LS writes its activity log and trace information. DFHJS2LS creates the file, but not the directory structure, if it does not exist.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHJS2LS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHJS2LS uses when generating the web service binding file and language structure. You can select these options:

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 Use this mapping level to generate JSON schema using the full set of options available.

4.0 Use this mapping level with a CICS TS 5.2 region when you want to use UTF-16.

For more information about mapping levels, see Mapping levels for the CICS assistants

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|LESS-DUP-NAMES}

Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

LESS-DUP-NAMES

This parameter generates non-structural structure field names with `_value` at the end of the name to enable direct referencing to the field. For example, in the following PLI language structure, when MAPPING-OVERRIDES=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with `_value`:

```
09 streetName,
    12 streetName                    CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

The resulting structure is as follows:

```
09 streetName,
    12 streetName_value              CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

SAME-AS-MAPPING-LEVEL

This parameter generates language structures in the same style as the mapping level. This is the default.

UNDERSCORES-AS-HYPHENS

For COBOL only. This parameter converts any underscores in the JSON schema to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure they are unique. For more information, see JSON schema to COBOL mapping in Developing applications.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}

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

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

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 The generated web service binding file deploys into a CICS TS 4.1 region or later.

Note: JSON support is only available from CICS TS 4.2 onwards.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

NAME-TRUNCATION={LEFT|RIGHT}
Specifies whether JSON names are truncated from the left or the right. The CICS web services assistant truncates JSON names to the appropriate length for the high-level language specified; by default names are truncated from the right.

PDSCP=*value*
Specifies the code page that is used in the partitioned data set members that are 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, the z/OS UNIX System Services code page is used. For example, you might specify **PDSCP**=037.

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

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.

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of a CICS PROGRAM resource.

When DFHJS2LS is used to generate a web service binding file that is used in a service provider, you must supply this parameter. It specifies the resource name of the application program that is exposed as a web service.

When DFHJS2LS is used to generate a web service binding file that is used in a service requester, omit this parameter.

REQMEM=*value*
Specifies a one to six character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contains the high-level language structures for the web service request, which is the input data to the application program.

DFHJS2LS generates the name of partitioned data set member by appending 01 to the prefix.

RESPMEM=*value*
Specifies a one to six character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contains the high-level language structures for the web service response, which is the output data from the application program.

DFHJS2LS generates the name of partitioned data set member by appending 01 to the prefix.

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 *request01* and *response01*.
If one or both names are 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 sync point.

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

YES The remote web service can issue a sync point. 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 one to four 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 by 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 uses to access the web service. CICS uses the value that is specified when it generates a URIMAP resource from the web service binding file that is created by DFHJS2LS. 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 one to eight 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ #

WIDE-COMP3={**NO**|**YES**}
For COBOL only. Controls the maximum size of the packed decimal variable length in the COBOL language structure.

NO DFHJS2LS limits the packed decimal variable length to 18 when generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.

YES DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

WSBIND=*value*
The fully qualified z/OS UNIX name of the web service binding file. DFHJS2LS creates the file, but not the directory structure, if it does not exist. The file extension defaults to **.wsbind**.

Other information

- The user ID under which DFHJS2LS runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

Example

```
//JS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
JSON-SCHEMA-REQUEST=example.json
JSON-SCHEMA-RESPONSE=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

DFHJS2LS: JSON schema to high-level language conversion for RESTful services:

The DFHJS2LS procedure generates a high-level language data structure and a web service binding file from a JSON schema. You can use DFHJS2LS when you prepare to create a RESTful JSON service provider.

The job control statements for DFHJS2LS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

The DFHJS2LS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

The relevant usage mode for the DFHLS2JS or DFHJS2LS procedure depends on your requirements:

- DFHLS2JS: High-level language to JSON schema conversion for linkable interface
- DFHJS2LS: JSON schema to high-level language conversion for linkable interface
- DFHLS2JS: High-level language to JSON schema conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for RESTful services

Job control statements for DFHJS2LS

JOB Starts the job.

EXEC Specifies the procedure name (DFHJS2LS).

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 DFHJS2LS:

JAVADIR=*path*
Specifies the name of the Java directory that is used by DFHJS2LS. The value of this parameter is appended to `/usr/lpp/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 a prefix that extends the z/OS UNIX directory path that is used on other parameters, or `''` (empty string) if no prefix is used.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHPREF** parameter.

SERVICE=*value*
Use this parameter only when directed to do so by IBM Support.

TMPDIR=*tmpdir*
Specifies the location of a directory in z/OS UNIX that DFHJS2LS uses as a temporary workspace. 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 DFHJS2LS uses to construct the names of the temporary workspace files.

The default value is `JS2LS`.

USSDIR=*path*
Specifies the name of the CICS TS directory in the UNIX System Services file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*. This must be specified as `'.'` (period) if the default is used.

Typically, you do not 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

DFHJS2LS creates the following three temporary files at run time:

tmpdir`/`*tmpprefix*`.in`

tmpdir`/`*tmpprefix*`.out`

tmpdir`/`*tmpprefix*`.err`

where:

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

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

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

`/tmp/JS2LS.in`

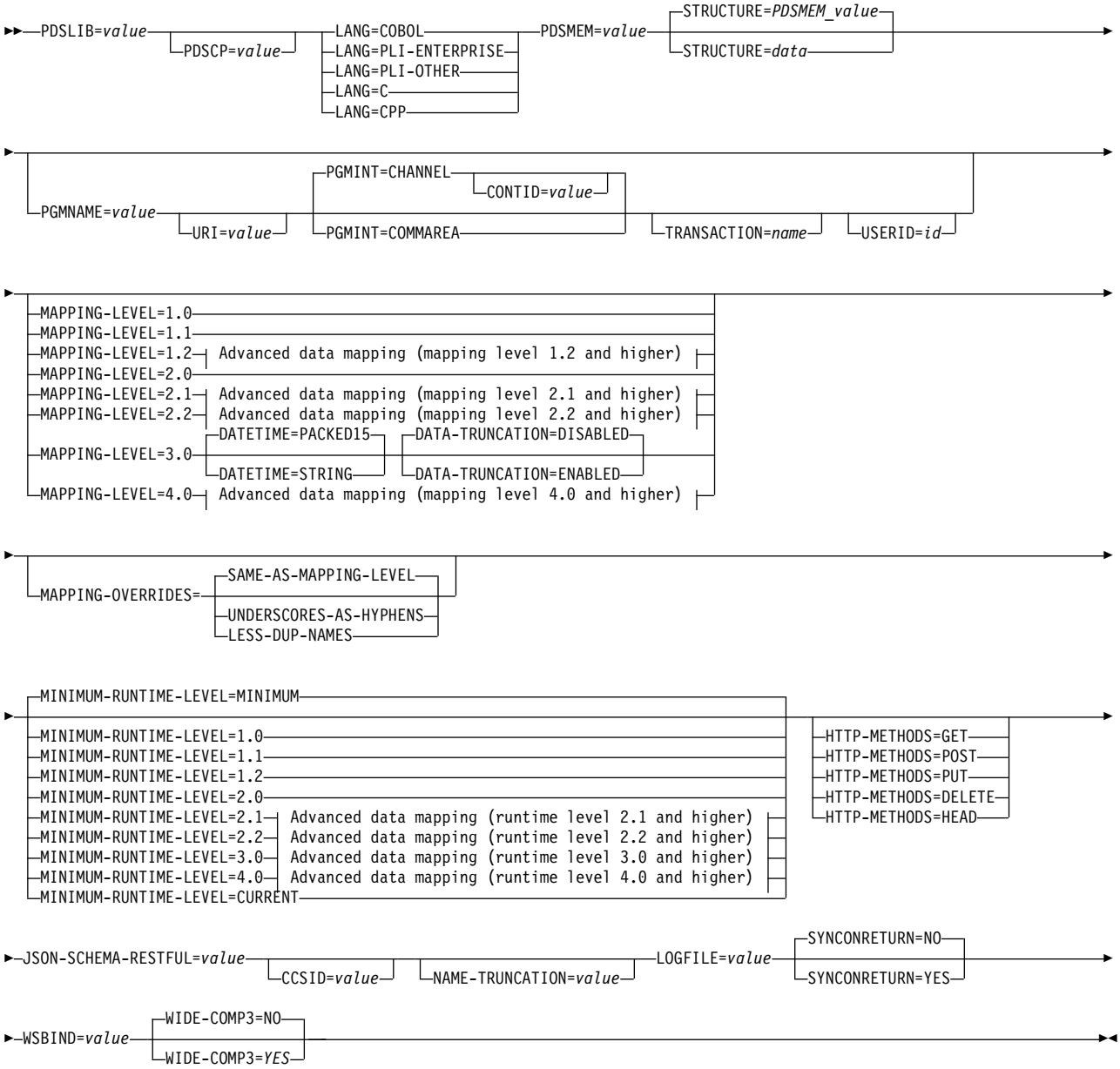
`/tmp/JS2LS.out`

```
/tmp/JS2LS.err
```

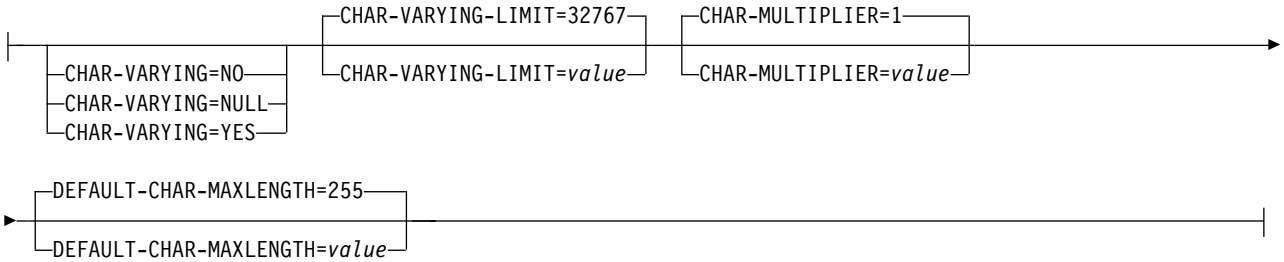
Important: DFHJS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHJS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that 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 DFHJS2LS



Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):

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 - 80 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 Unicode Services User's Guide and Reference. If you do not specify this parameter, the application data structure is encoded by using the CCSID specified in the system initialization parameter.

CHAR-MULTIPLIER={**1**|*value*}
: Specifies the number of bytes to allow for each character when the mapping level is 1.2 or later. The *value* of this parameter can be a positive integer in the range of 1 - 2,147,483,647. 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 might 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.

When you set **CCSID=1200** (indicating UTF-16), the only valid values for **CHAR-MULTIPLIER** are 2 or 4. When you use UTF-16, the default value is 2. Use **CHAR-MULTIPLIER=2** when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use **CHAR-MULTIPLIER=4** when you expect application data to contain characters that require 2 UTF-16 encoding units.

Note: Setting **CHAR-MULTIPLIER** to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger **CHAR-MULTIPLIER** value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.

CHAR-VARYING=NO|NULL|YES
Specifies how variable-length character data is mapped when the mapping level is 1.2 or higher. Variable-length binary data types are always mapped to either a container or a varying structure. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

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 two related elements: data-length and the data.

CHAR-VARYING-LIMIT=32767|*value*
Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure when the mapping level is 1.2 or higher. If the character or binary 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 32,767 bytes.

CONTID=*value*
In a service provider, specifies the name of the container that holds the top-level data structure that is used to represent a JSON message.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={DISABLED|ENABLED}
Specifies if variable length data is tolerated in a fixed-length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME=PACKED15|STRING
Specifies how JSON date-time elements are mapped to the language structure.

PACKED15
The default is that any JSON date-time element is processed as a timestamp and is mapped to CICS ABSTIME format.

STRING
The JSON date-time element is processed as text.

DEFAULT-CHAR-MAXLENGTH=255|*value*
Specifies the default array length of character data in characters for mappings where no length is implied in the web service description document, when the mapping level is 1.2 or higher. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647.

HTTP-METHODS={*GET*|*POST*|*PUT*|*DELETE*|*HEAD*},{*GET*|*POST*|*PUT*|*DELETE*|*HEAD*},*
This is an optional parameter.

The default value is for GET, POST, PUT, and DELETE to be set, which tells DFHJS2LS that the application supports the four main RESTful operations.
If a value is provided, DFHJS2LS builds a WSBind file in which only the explicitly specified HTTP methods are accepted.
If an application wants to implement the HEAD method, it must deliberately opt-in to doing so. By default DFHJS2LS assumes that the application does not support incoming HTTP HEAD methods.
If a JSON client attempts to use an unsupported HTTP method, CICS responds with an HTTP 405 response.

INLINE-MAXOCCURS-LIMIT=1|*value*
Specifies whether inline variable repeating content is used based on the `maxItems` JSON schema keyword. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.

The **INLINE-MAXOCCURS-LIMIT** parameter is available only at mapping level 2.1 onwards. The value of **INLINE-MAXOCCURS-LIMIT** can be a positive integer in the range of 0 - 32,767. A value of `0` indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the *value* of the `maxOccurs` attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise inline mapping is used.

When deciding whether you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.

JSON-SCHEMA-RESTFUL=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the response JSON schema is stored.

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 z/OS UNIX name of the file into which DFHJS2LS writes its activity log and trace information. DFHJS2LS creates the file, but not the directory structure, if it does not exist.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHJS2LS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHJS2LS uses when generating the web service binding file and language structure. You can select these options:

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 Use this mapping level to generate JSON schema using the full set of options available.

4.0 Use this mapping level with a CICS TS 5.2 region when you want to use UTF-16.

For more information about mapping levels, see Mapping levels for the CICS JSON assistants in Developing applications.

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|LESS-DUP-NAMES}
Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

LESS-DUP-NAMES
This parameter generates non-structural structure field names with `_value` at the end of the name to enable direct referencing to the field. For example, in the following PLI language structure, when MAPPING-OVERRIDES=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with `_value`:

```
09 streetName,
    12 streetName                    CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

The resulting structure is as follows:

```
09 streetName,
    12 streetName_value              CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

SAME-AS-MAPPING-LEVEL
This parameter generates language structures in the same style as the mapping level. This is the default.

UNDERSCORES-AS-HYPHENS
For COBOL only. This parameter converts any underscores in the JSON schema to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure they are unique. For more information, see JSON schema to COBOL mapping in Developing applications.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}
Specifies the minimum CICS runtime environment into which the web service binding file can be deployed. If you select a level that does not match the other parameters that you specified, you receive an error message. You can select these options:

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

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 The generated web service binding file deploys into a CICS TS 4.1 region or later.

Note: JSON support is only available from CICS TS 4.2 onwards.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

NAME-TRUNCATION={LEFT|RIGHT}
Specifies whether JSON names are truncated from the left or the right. The CICS web services assistant truncates JSON names to the appropriate length for the high-level language specified; by default names are truncated from the right.

PDSCP=*value*
Specifies the code page that is used in the partitioned data set members that are specified in the **PDSMEM** parameter, where *value* is a CCSID number or a Java code page number. If this parameter is not specified, the z/OS UNIX System Services code page is used. For example, you might specify **PDSCP**=037.

PDSLIB=*value*
Specifies the name of the partitioned data set that contains the generated high-level language. The data set members that are used for the request and response are specified in the **PDSMEM** parameter.

PDSMEM=*value*
Specifies the 1-6 character prefix that DFHJS2LS uses to generate the name of the partitioned data set member that will contain the high-level language structures.

DFHJS2LS generates a partitioned data set member by appending 01 to the prefix.

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.

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of a CICS PROGRAM resource.

When DFHJS2LS is used to generate a web service binding file that is used in a service provider, you must supply this parameter. It specifies the resource name of the application program that is exposed as a web service.

When DFHJS2LS is used to generate a web service binding file that is used in a service requester, omit this parameter.

STRUCTURE=*name*
For C and C++ only, specifies how the name of the structure is generated.

The generated structure is given the name of *name01* .
If the name is omitted, the structure has the same name as the partitioned data set member name generated from the **PDSMEM** parameter that you specify.

SYNCONRETURN={**NO**|**YES**}
Specifies whether the remote web service can issue a sync point.

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

YES The remote web service can issue a sync point. 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 one to four 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 by 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 uses to access the web service. CICS uses the value that is specified when it generates a URIMAP resource from the web service binding file that is created by DFHJS2LS. The parameter specifies the path component of the URI to which the URIMAP definition applies. When using wildcards * at the end of a URI, the URI value must be followed by a comma.

USERID=*id*
In a service provider, this parameter specifies a one to eight 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ #

WIDE-COMP3=**NO**|**YES**
For COBOL only. Controls the maximum size of the packed decimal variable length in the COBOL language structure.

NO DFHJS2LS limits the packed decimal variable length to 18 when

generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.

YES DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

WSBIND=*value*

The fully qualified z/OS UNIX name of the web service binding file. DFHJS2LS creates the file, but not the directory structure, if it does not exist. The file extension defaults to `.wsbind`.

Other information

- The user ID under which DFHJS2LS runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

Example

```
//JS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA-RESTFUL=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example/*,
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

Creating a JSON service provider application by using the JSON assistant

You can create a service provider application from a JSON schema that complies with JSON schema v4 (draft), or from a high-level language data structure. The CICS JSON assistant helps you to deploy your CICS applications in a service provider setting.

About this task

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

- Start with a JSON schema and use the assistant to generate the language data structures.

  Use this option when you want to implement a service provider that conforms with an existing web service description.

- Start with the language data structures and use the assistant to generate the JSON schema.

  Use this option when you want to expose an existing program as a JSON service.

Related information:

JSON schema v4 (draft)

Creating a service provider application from a JSON schema:

Using the CICS JSON assistant, you can create a service provider application from a JSON schema.

Before you begin

Before you can create a service provider application, the following conditions must be satisfied:

- Your web services description must be in a UNIX file in z/OS and you must create a suitable provider mode pipeline in the CICS region.
- You must define to OMVS the user ID under which DFHJS2LS runs.
- The user ID must have read permission to z/OS UNIX and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **JSON-SCHEMA-REQUEST** , **JSON-SCHEMA-RESPONSE**, and **JSON-SCHEMA-RESTFUL** parameters.
- You must allocate sufficient storage to the user ID for the ID to run Java. You can use any supported version of Java. By default, DFHJS2LS uses the Java version specified in the **JAVADIR** parameter.

About this task

You can use the JSON assistant to create language structures from your JSON schema for the service provider application.

Procedure

1. Use the DFHJS2LS batch program to generate a web service binding file and one or more language data structures. DFHJS2LS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider these options when you enable an existing application for web services:
   - Which mechanism will CICS use to pass data to the service provider application program? You can use channels and pass the data in containers or use a COMMAREA. Channels and containers are recommended. Specify them with the **PGMINT** parameter.
   - Which language do you want to generate? DFHJS2LS can generate COBOL, C/C++, or PL/I language data structures. Specify the language using the **LANG** parameter.
   - Which mapping level do you want to use? The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to use the highest level of mapping available. Specify the mapping level with the **MAPPING-LEVEL** parameter.

- Which URI do you want the web service requester to use? Specify a relative URI using the **URI** parameter; for example, `URI=/my/test/webservice`. The value is used by CICS when it creates the URIMAP resource.
- Under which transaction and user ID will you run the web service request and response? You can use an alias transaction to run the application to compose a response to the service requester. The alias transaction is attached under the user ID. Specify it with the **TRANSACTION** and **USERID** parameters. These values are used when creating the URIMAP resource. If you do not want to use a specific transaction, do not use these parameters.

When you submit DFHJS2LS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The language structures are placed in the partitioned data set that you specified with the **PDSLIB** parameter.

2. Copy the generated web service binding file to the `pickup` directory of the provider mode PIPELINE resource that you want to use for your web service application. You must copy the binding file in binary mode.
3. Write a service provider application program to interface with the generated language structures and implement the required business logic.
4. Use the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and a URIMAP resources.
   - The WEBSERVICE resource encapsulates the web service binding file in CICS and is used at run time.
   - The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.

   Alternatively, you can define the resources yourself, although this is not recommended.

Results

If you have any problems submitting DFHJS2LS, or the resources do not install correctly, see Troubleshooting the JSON assistant in Troubleshooting.

JSON schema v4

JSON schema validation v0

Creating a service provider application from a data structure:

Use the CICS web services assistant to create a service provider application from a high-level language data structure.

Before you begin

Before you create a service provider application, make sure that your setup complies with these preconditions:

- Your high-level language data structures must meet the following criteria:
  - The data structures must be defined separately from the source program; for example, in a COBOL copybook.
  - If your PL/I or COBOL application program uses different data structures for input and output, the data structures must be defined in two different members in a partitioned data set. If the same structure is used for input and output, the structure must be defined in a single member.

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

- The language structures must be available in a partitioned data set and you must create a suitable PIPELINE resource in the CICS region.
- You must define to OMVS the user ID that DFHLS2JS uses to run.
- The user ID must have read permission to z/OS UNIX and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** output parameters.
- The user ID must have a sufficiently large storage allocation to run Java. You can use any supported version of Java. By default, DFHLS2JS uses the Java version that is specified in the **JAVADIR** parameter.

Procedure

Follow these steps to create a service provider application from a high-level data structure:

1. If the service provider application interface uses channels and many containers, create a channel description document that describes the interface in JSON. You must save the channel description document in a suitable directory on z/OS UNIX. CICS uses this document to construct and deconstruct a JSON message from the containers on a channel. Alternatively, you can use one container in a channel and not create a channel description document.

   For more information about how to create a channel description document, see "Creating a channel description document" on page 187.
2. Use the DFHLS2JS batch program to generate a web service binding file and web service description from the language structure. The DFHLS2JS batch program can be found in *HLQ*.XDFHINST where *HLQ* is the location where you installed CICS. DFHLS2JS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider the following options when enabling web services for an existing application:
   - Which mechanism do you want CICS to use to pass data to the service provider application program? You can use channels and pass the data in containers, or use a COMMAREA. Specify the mechanism by using the **PGMINT** parameter. If your application interface uses channels and many containers, specify the **REQUEST-CHANNEL** parameter and optionally the **RESPONSE-CHANNEL**. You can use these parameters only when the mapping level is 3.0 or higher.
   - Which mapping level do you want to use? The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You must specify the highest level of mapping available in the **MAPPING-LEVEL** parameter.
   - Which URI do you want the web service to use? Specify an absolute URI by using the **URI** parameter; for example, **URI**=http://www.example.org:80/my/test/webservice. The relative part of this address, /my/test/webservice, is used when creating the URIMAP resource.

   When you submit DFHLS2JS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The generated JSON schemas are placed in the location that you specified with the **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** parameters.
3. Review the generated JSON schema.

These schemas are used to define the input and output data formats for the JSON web service. The application developer must use these schemas when creating an application to interact with the JSON web service.

Note: Changing the generated schema invalidates the generated web service binding file, `WSBind`.

If you want to change the schema, for example, to rename the fields within the schema, you must use DFHJS2LS to generate a new web service binding file, and a new set of language structures. The application program in CICS must be changed to use the new language structures.

4. Copy the web service binding file to the pickup directory of the provider mode pipeline that you want to use for your web service application. You must copy the web service binding file in binary mode.
5. Use the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and a URIMAP resource.
   - The WEBSERVICE resource contains the web service binding file in CICS and is used at run time.
   - The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.

   Alternatively, you can define the resources yourself.

Results

The creation of your service provider application is complete.

If you have any problems submitting DFHLS2JS, or the resources do not install correctly, see Troubleshooting the JSON assistant in Troubleshooting.

What to do next

Make the web services description available to anyone who develops a web service that will access your service.

Creating a channel description document:

Create a channel description document when your service provider application uses a channel interface with many containers.

About this task

Use an XML editor to create the channel description document. The schema for the channel description is called `channel.xsd` and is in the `/usr/lpp/cicsts/cicsts53/schemas/channel` directory (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX).

Procedure

1. Create an XML document with a <channel> element and the CICS channel namespace:

   ```
   <channel name="myChannel" xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
   </channel>
   ```

2. Add a <container> element for every container that the application program interface uses on the channel. You must use name, type and use attributes to describe each container. The following example shows six containers with different attribute values:

```
<container name="cont1" type="char" use="required"/>
<container name="cont2" type="char" use="optional"/>
<container name="cont3" type="bit" use="required"/>
<container name="cont4" type="bit" use="optional"/>
<container name="cont5" type="bit" use="required">
    <structure location="//HLQ.PDSNAME(MEMBER)"/>
</container>
<container name="cont6" type="bit" use="optional">
    <structure location="//HLQ.PDSNAME(MEMBER2)"/>
</container>
```

The structure element indicates that the content is defined in a language structure located in a partitioned data set member.

3. Save the XML document in z/OS UNIX.

Channel schema

The channel description document must conform to the following schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
   xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel" elementFormDefault="qualified">
   <element name="channel"> 1
      <complexType>
         <sequence>
            <element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0"> 2
               <complexType>
                  <sequence>
                     <element name="structure" minOccurs="0"> 3
                       <complexType>
                           <attribute name="location" type="string" use="required"/>
                           <attribute name="structure" type="string" use="optional"/>
                       </complexType>
                     </element>
                  </sequence>
                  <attribute name="name" type="tns:name16Type" use="required"/>
                  <attribute name="type" type="tns:typeType" use="required"/>
                  <attribute name="use" type="tns:useType" use="required"/>
               </complexType>
             </element>
          </sequence>
          <attribute name="name" type="tns:name16Type" use="optional" />
       </complexType>
   </element>
  <simpleType name="name16Type">
      <restriction base="string">
         <maxLength value="16"/>
      </restriction>
  </simpleType>
  <simpleType name="typeType">
      <restriction base="string">
        <enumeration value="char"/>
        <enumeration value="bit"/>
      </restriction>
  </simpleType>
  <simpleType name="useType">
     <restriction base="string">
        <enumeration value="required"/>
        <enumeration value="optional"/>
     </restriction>
  </simpleType>
</schema>
```

1. This element represents a CICS channel.
2. This element represents a CICS container within the channel.

3. A structure can only be used with 'bit' mode containers. The 'location' attribute indicates the location of a file that maps the contents of container. The 'structure' attribute may be used in C and C++ to indicate the name of structure.

What to do next

Run DFHLS2JS to create the mappings and JSON schema for the web service provider application. DFHLS2JS puts the mappings for the channel in the JSON schema in the order that the containers are specified in the channel description document.

Customizing generated JSON schemas:

The JSON schemas that are generated by DFHLS2JS contain some automatically generated content that might be appropriate for you to change before publishing. Customizing JSON schemas can result in regenerating the web services binding file and, in some cases, writing a wrapper program.

About this task

Follow these steps to customize generated JSON schemas:

Procedure

1. To advertise support for HTTPS, use the **URI** parameter in DFHLS2JS to set an absolute URI.
2. To supply the network location of your web service, use the **URI** parameter in DFHLS2JS to set an absolute URI.
3. Consider whether the automatically generated names in the JSON schema are appropriate for your purposes. You can rename the properties.

   These values form part of the programmatic interface to which you code a client program. If the generated names are not sufficiently meaningful, maintenance of your application code might be more difficult over a long period of time.

   If you change any of these values, you must use DFHJS2LS to regenerate the web services binding file. The language structures that are produced are unlikely to be compatible with your existing application, so application changes might be required. Review the generated language structures, and consider writing a wrapper program as discussed in step 4.
4. Consider if the COMMAREA fields exposed in the JSON schemas are appropriate. You might consider removing any fields that are not helpful to a JSON client developer:
   - Fields that are used only for output values can be removed from the schema that maps the input data structures.
   - Filler fields.
   - Automatically generated annotations.

   If you make any of these changes, you must regenerate the web services binding file using DFHJS2LS. The new language structures that are generated are not compatible with the original language structures, so you must write a wrapper program to map data from the new representation to the old one. This wrapper program needs to perform an **EXEC CICS LINK** command 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 JSON client developers.

Results

You have a customized JSON schema that matches your business requirements, and a PROGRAM in CICS that implements them.

Creating a RESTful web service provider application:

The CICS implementation of RESTful JSON web services is similar to that of SOAP web services. Most of the concepts and architecture are shared, but CICS requires the use of a JSON schema.

About this task

Implementing a RESTful JSON web service involves the following tasks:

Procedure

1. Generate the application interface.

   Input:

   - The JSON schema that defines the data model for the RESTful web service.

   Output:

   - The language structures (for example, COBOL copy book) that map the JSON schema.
   - A WSBind file.

   Run the DFHJS2LS utility and specify the appropriate input parameters. These parameters include:

   - The location of the JSON schema.
   - The list of supported methods (GET, PUT, POST, and DELETE are enabled by default).
   - The URI at which the service is deployed.
   - The name of the application PROGRAM that implements the service.
   - Any required data-mapping parameters.
2. Create an application that uses this interface.

   Input:

   - The language structures from step 1.
   - An awareness of which RESTful operations will be implemented

   Output:

   - A program suitable for deployment to CICS

   Write a program that does the following:

   Note: If you supply the name of your own container on the **CONTID** parameter to DFHJS2LS, you must use this container instead of DFHWS-DATA whenever it is mentioned in the following steps.

   a. Examine the URI to understand the identity of the resource. CICS provides several containers to help you identify interesting components of the URI. The containers are described in Table 2 on page 191. The examples show the contents of each container for the URI:

      ```
      http://www.example.org:10000/JSONServices/CustomerDetails/13388?action=query
      ```

If the URIMAP that matched has PATH of `/JSONServices/CustomerDetails/*`

Table 2. DFHWS-URI containers. DFHWS-URI containers

Container	Contents	Example
DFHWS-URI in Configuring	The complete URI	`http://www.example.org:10000/JSONServices/CustomerDetails/13388?action=query`
	The portion of URI path that matched the URIMAP	`/JSONServices/CustomerDetails/*`
DFHWS-URI-RESID in Configuring	The URI path with the portion matched by the URIMAP removed (*the resource identifier*)	`13388`
DFHWS-URI-QUERY in Configuring	The query string	`action=query`

b. Validate the URI. If there is a problem, report the problem to CICS and terminate.

c. Query the DFHHTTPMETHOD container to determine which method is being driven. For more information, see DFHHTTPMETHOD in Configuring.

d. For a POST (create) method (if needed):
   - Read the input data from the DFHWS-DATA container.
   - Interpret the data by using the language structure(s) generated in step 1 on page 190.
   - Validate the data. If there is a problem, report the problem to CICS and terminate.
   - Perform whatever application-specific processing is required to create the *resource*.
   - Optionally, write to the DFHRESPONSE container to notify the client of the identifier of the new resource. The contents of this container are not transformed by CICS, but sent directly in the HTTP response. The DFHWS-DATA container is ignored.

e. If a GET (inquire) or HEAD (inquire) method is needed, write the data that represents the *resource* to the DFHWS-DATA container.

f. If a PUT (set) method is needed:
   - Read the input data from the DFHWS-DATA container.
   - Interpret the data by using the language structure(s) generated in step 1 on page 190.
   - Validate the data. If there is a problem, report the problem to CICS and terminate.
   - Perform whatever application-specific processing is required to update the *resource*.

g. If a DELETE method is needed, perform whatever application-specific processing is required to delete the *resource*.

Note: The RESTful data model that is implemented by CICS does not send a response body for the PUT, POST, or DELETE methods by default. RESTful applications typically use the HTTP status code to indicate success or failure. If

the application completes normally, CICS sends an HTTP response of 200 (OK). For more information about sending error responses, see Application error reporting in Developing applications. If you want to send a response body for a PUT, POST or DELETE method, you must write the DFHRESPONSE container. If present, CICS sends the contents of this container in the HTTP body without further processing. CICS ignores the DFHWS-DATA container in the response processing for these methods.

3. Deploy the artifacts.

   Input:

   - The WSBind file from step 1 on page 190.
   - The CICS Program from step 2 on page 190.
   - A CICS provider mode pipeline resource that is configured for JSON.

   Output:

   - A deployed RESTful JSON web service.

   Deploy the program to CICS in the normal way.

   Either:

   - Deploy the WSBind file to the Pipeline's 'pickup' directory; then issue a **PIPELINE SCAN** command to create the WEBSERVICE and URIMAP resources.
   - Manually define and install a WEBSERVICE resource and associated URIMAP resource. The URIMAP must associate the URI with both the PIPELINE and the WEBSERVICE.

4. Test the service.

   Input:

   - The JSON schema.
   - The URI of the RESTful web service.
   - The deployed service from step 3.
   - A JSON test client of your choice.

   Output:

   - A successfully handled request.

   Use the test client of your choice to send a JSON request to CICS.

   If you receive an unexpected response, attempt problem determination. For more information, see Troubleshooting problems with JSON requests.

Related information:

RESTful services

Design considerations for RESTful web service provider applications:

This topic describes some issues you should consider when planning and designing a RESTful web service provider application for JSON.

Collections of Resources

A common design for RESTful APIs is to support the retrieval of collections of resources. For example, a Service might exist that returns a set of objects as follows:

```
GET /Services/CustomerDetails?Surname=Cooper
```

This request is expected to return information on all CustomerDetails objects where the Surname is "Cooper". Individual CustomerDetails objects may be returned using a more specific URI such as:

```
GET /Services/CustomerDetails/Customer27
```

In this example `Customer27` is the primary key for a specific customer. The output from this second query will be an instance of the CustomerDetails object. The output from the first query is less clear: it could return a list of CustomerDetails objects, or it could return a list of URIs for CustomerDetails objects (which the client can go on to retrieve individually). Both conventions are common.

To implement a Collection in CICS, create a JSON schema that describes either a list of data instances, or a list of URIs. You can then build the Service and implement it as normal. In this example you might choose to only implement the GET method. You could consider implementing a pagination Service to allow a client to page backwards and forwards through large data sets, for example:

```
GET /Services/CustomerDetails?startRecord=200&endRecord=225
```

You're likely to need two URIMAP resources in CICS (and two WEBSERVICE resources). One that maps the root URI structure for the Collection, and a wild-carded URIMAP for the Instances. For example:

```
URIMAP1: Path=/Services/CustomerDetails  WEBSERVICE=CollectionService
URIMAP2: Path=/Services/CustomerDetails/* WEBSERVICE=InstanceService
```

Cache Management

The RESTful architecture encourages integration with standard HTTP cache management techniques. This allows the results of GET requests to be cached in the network, thereby reducing the burden on the server. The mechanism for doing this involves setting an expiration date/time for data returned for GET requests.

There is no general purpose mechanism to support application controlled cache expiration in CICS, but a Pipeline Handler program could be written to add the appropriate HTTP Header using the EXEC CICS WEB WRITE HTTPHEADER API. Application programs can do something similar, but only if they are hosted in the same CICS region that receives the HTTP request.

Application error reporting:

Read this topic to understand how RESTful JSON web service provider applications can report errors to clients.

In top-down scenarios, it is likely that the application is required to report error conditions. For example, an error might be: "`Account Number not recognized`". This requirement is unique to top-down scenarios; in bottom-up development the application either has an error reporting mechanism that is encoded in the data fields, or it abends. In the top-down scenario, the JSON schema is unlikely to define a field in which to report errors, so an alternative is needed.

For SOAP-based web services, this problem is addressed by using the **EXEC CICS SOAPFAULT** API. SOAP Fault messages do not exist in JSON. Instead, you can use the DFHHTTPSTATUS container to report application detected errors for JSON applications. For more information, see DFHHTTPSTATUS in Configuring.

Note: Applications might also use the DFHRESPONSE container, and other control containers, to provide a more detailed error response, if they want to do so.

JSON web service restrictions

Use this reference material to understand capabilities that are not supported by JSON web services.

The following capabilities are not supported:

- Requester mode pipelines with JSON are not supported.
- Runtime validation of JSON data against schema is not supported. The value of the VALIDATION attribute of a WEBSERVICE resource that is used with a JSON payload is ignored.
- Use of namespaces in JSON data (Badgerfish or Mapped conventions) is not supported.
- JSON payloads sent to CICS must be encoded in UTF-8. No other encoding is supported. Similarly, JSON sent by CICS is always encoded in UTF-8.
- WebSphere MQ transports with JSON pipelines are not supported.
- Vendor transformer programs are not supported for use with the JSON transformer.
- Reuse of WSBind files that are created for SOAP web services applications in a JSON pipeline is not supported. WSBind files for use with JSON service provider applications must be generated by the JSON assistant.
- If a JSON payload is missing some of its required content when CICS transforms it, the equivalent fields within the data structures are not initialized when passed to the application program.
- CICS cannot transform integer values greater than the maximum value for a signed long (2^63 - 1) unless they are enclosed within quotes.
- Use of simple data types or arrays is not supported at the root of a JSON Schema. The JSON Schema is required to describe a JSON Object, though the JSON Object can be composed of simple data types and arrays.
- If an array is declared in a JSON schema with a maxItems value of 1, CICS serializes the array as a simple string or integer when generating JSON at runtime.

Important: The only supported characters for JSON property names are: `A-Z a-z _ :` for the first character and `A-Z a-z 0-9 _ : . -` for all subsequent characters.

The Axis2 web services support today has a number of options for development and deployment of applications and customizations. The following options are not supported:

- User-supplied application handlers - you must use the CICS supplied application handler class `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler`.
- User-written Axis2 Java applications.
- The SOAPFAULT and WS-Addressing APIs cannot be used with the JSON pipeline.

Container restrictions

Note that some pipeline containers are not populated when processing a JSON request. For more information, see Containers used in the pipeline in Configuring.

Differences in RESTful web services

On INQUIRE PIPELINE:

- SOAPLEVEL returns NOTSOAP

- The MTOMNOXOPST, MTOMST, SENDMTOMST, SOAPRNUM, SOAPVNUM, XOPDIRECTST, and XOPSUPPORTST attributes are not used.

On INQUIRE WEBSERVICE:

- The ARCHIVEFILE, BINDING, VALIDATIONST, XOPDIRECTST, and XOPSUPPORTST attributes are not used.
- WSDLFILE returns the name of the JSON schema file that is associated with the WEBSERVICE.

On the WEBSERVICE resource:

- The ARCHIVEFILE and VALIDATION parameters are not used and their values are ignored.
- WSDLFILE is the name of the JSON schema file that is associated with the WEBSERVICE.

CICS resources for web services

PIPELINE, WEBSERVICE, URIMAP and TCPIPSERVICE resources support web services in CICS.

PIPELINE

A PIPELINE resource definition is required for every web service. It provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE resource definition defines an infrastructure that can be used by many applications. The information about the message handlers is supplied indirectly: the PIPELINE resource definition specifies the name of a z/OS UNIX file that 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 definitions 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 runtime 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 pickup directory for the PIPELINE resource definition is scanned. This can occur 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 resource are distinguished by the PROGRAM attribute in the resource definition: for a service provider, the attribute must be specified; for a service requester, it must be omitted.

URIMAP

A URIMAP definition is required in a service provider when it contains information that maps the URI of an inbound web service request to the other resources (such as the PIPELINE resource) that will service the request. This URIMAP definition is also required if you are using HTTP basic authentication, because the URIMAP resource definition specifies that the service requester user ID information is passed in an HTTP authorization header to the service provider.

A second optional URIMAP definition can exist in a service provider for WSDL discovery. This URIMAP resource definition contains information that maps the URI of an inbound request for the WSDL document or documents associated with the web service.

For service providers deployed using the CICS web services assistant, although CICS provides the usual resource definition mechanisms, the URIMAP resources are typically created automatically when the pick directory is scanned. This scan occurs when the PIPELINE resource is installed or as a result of a PERFORM PIPELINE SCAN command. The URIMAP resource that provides CICS with the information to associate the WEBSERVICE resource with a specific URI is a required resource. The attributes for this resource are specified by a web service binding file in the pickup directory. The URIMAP resource that provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI is an optional resource and is created if either a WSDL file or WSDL archive file are present in the pickup directory. For more information about creating URIMAP resources for web service providers, see "Creating a web service provider by using the web services assistant" on page 128.

For service requesters, CICS does not create any URIMAP resources automatically when the PIPELINE resource is installed or as a result of a PERFORM PIPELINE SCAN command. Service requesters are not required to use URIMAP resources when they make requests; they can specify the URI of the outbound request directly in the application program. However, if you create a URIMAP resource for the client request, and your service requesters use the URIMAP resource to provide the URI, you gain these advantages:

- System administrators can manage any changes to the endpoint of the connection, so you do not need to recompile your applications if the URI of a service provider changes.
- You can choose to make CICS keep the connections that were opened with the URIMAP resource open after use, and place them in a pool for reuse by the application for subsequent requests, or by another application that calls the same service. Connection pooling is only available when you specify a URIMAP resource that has the SOCKETCLOSE attribute set. For more information about the performance benefits of connection pooling, see Connection pooling for HTTP client performance in Improving performance.

Configuring URIMAP resource attributes in a certain way might enable inbound requests being processed by directly attached user transactions, and bypassing the web attach task. For more information, see HTTP requests are processed by directly attached user transactions in Improving performance.

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

The resources that are required to support a particular application program depend on the following criteria:

- Whether the application program is a service provider 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
Provider	No	Yes	No	Yes	See note 2
Requester	Yes	Yes	Yes	See note 3	No
Requester	No	Yes	No	3	No

Notes:

1. When the CICS web services assistant is used to deploy an application program, the WEBSERVICE and two URIMAP resources can be created automatically when the pickup directory of the PIPELINE is scanned. The first URIMAP resource is required and provides CICS with the information to associate the WEBSERVICE resource with a specific URI. The second URIMAP resource is optional and provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI so that external requesters can use the URI to discover the WSDL archive file or WSDL document. The pickup directory of the PIPELINE scan occurs 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.
3. A URIMAP resource is optional for a service requester, and the CICS web services assistant does not generate one automatically. When you define your own URIMAP resources for service requesters to use, you can implement connection pooling, and manage changes to the URIs for service providers.

Configuring TCPIP resource attributes in a certain way might enable inbound requests being processed by directly attached user transactions, and bypassing the web attach task. For more information, see HTTP requests are processed by directly attached user transactions in Improving performance.

Typically, when you deploy many web services applications in a CICS system, you have more than one of each type of resource. In this case, you can share some resources between applications. Each web services file or resource is associated with one or more CICS resources of other types.

Table 3. Other CICS resources that are associated with each web services file and resource

Web services file or resource	Associated resources
Pipeline configuration file	• More than one PIPELINE resource that refers to the file.
PIPELINE	• More than one URIMAP resource that refers to the PIPELINE resource. • More than one WEBSERVICE resource that refers to the PIPELINE resource. • More than one web service binding file in the pickup directory of the PIPELINE resource.
Web service binding file	• One URIMAP resource that is automatically generated from the binding file. You can define further URIMAP resources for a service provider, and you can define URIMAP resources for a service requester. • One WEBSERVICE resource that is automatically generated from the binding file. You can define further WEBSERVICE resources if you need to.
WEBSERVICE	• More than one URIMAP resource. If the WEBSERVICE resource is automatically generated from the binding file for a service provider, CICS generates one corresponding URIMAP resource. You can define further URIMAP resources for a service provider, and you can define URIMAP resources for a service requester.
URIMAP	• Just one TCPIPSERVICE resource when it is explicitly named in the URIMAP resource.
TCPIPSERVICE	• Many URIMAP resources.

Web services discovery

WSDL documents associated with a Provider mode web service are automatically published to the Web.

A convention exists among web service hosting environments that allows the WSDL for a web service to be queried by a remote client (typically an Application Developer using a web browser) using the URI for the web service suffixed with `?wsdl`. This convention can make it easier to distribute WSDL to interested parties without the need for a formal WSDL repository. This convention is implemented in CICS.

For example, you might have a web service hosted in CICS and published under the following URI:

```
http://www.example.org:1234/example/WebService
```

The associated WSDL document could be recovered by requesting the following URI using a web browser:

```
http://www.example.org:1234/example/WebService?wsdl
```

WSDL documents for service providers can be published for discovery using URIMAP resources. When you install each PIPELINE resource, CICS scans the directory specified in the WSDIR attribute of the PIPELINE resource (the pickup directory). If this directory contains either a WSDL archive file or WSDL document, a second URIMAP resource is installed. This new URIMAP resource provides CICS with the information to associate the WSDL archive file or a WSDL document with a specific URI so that external requesters can use the URI to discover the WSDL archive file or WSDL document. This URI has the same path as the URI associated with the WEBSERVICE with the suffix `?wsdl` appended.

The WSDL archive file can contain one or more WSDL documents. If the pickup directory contains a WSDL archive file and a WSDL document, the URI returns only the WSDL archive. The archive file format that is supported is the .zip file type. It is also possible to discover the WSDL archive file or WSDL document using SPI and CEMT. The WSDL document in a WSDL archive file can be used for SOAP message validation.

Configuring CICS to use the WebSphere MQ transport

To use the WebSphere MQ transport with SOAP web services in CICS, you must configure your CICS region accordingly.

About this task

Note: You cannot use the Websphere MQ transport for JSON web services.

Procedure

1. Include the WebSphere MQ library *thlqual*.SCSQAUTH in the STEPLIB concatenation in your CICS procedure. Include the library after the CICS libraries to ensure that the correct code is used. *thlqual* is the high-level qualifier for the WebSphere MQ libraries.
2. Include the following WebSphere MQ libraries in the DFHRPL concatenation in your CICS procedure. Include the libraries after the CICS libraries to ensure that the correct code is used.

   thlqual.SCSQCICS

   thlqual.SCSQLOAD

   thlqual.SCSQAUTH

   thlqual is the high-level qualifier for the WebSphere MQ libraries. If you are using the CICS-WebSphere MQ API-crossing exit (CSQCAPX), also add the name of the library that contains the load module for the program. The SCSQCICS library is required only if you want to run WebSphere MQ supplied samples. Otherwise it can be removed from the CICS procedure.
3. Install an MQCONN resource for the CICS region. The MQCONN resource specifies the attributes of the connection between CICS and WebSphere MQ, including the name of the default WebSphere MQ queue manager or queue-sharing group for the connection. For more information, see Setting up an MQCONN resource in Configuring.
4. Specify the CICS system initialization parameter **MQCONN=YES** to start the CICS-WebSphere MQ connection automatically at CICS initialization.

   An MQCONN resource definition must be installed before CICS can start the connection to WebSphere MQ. When you start the connection automatically at CICS initialization, for an initial or cold start, the MQCONN resource definition must be present in one of the groups named in the list or lists named by the

GRPLIST system initialization parameter. For a warm or emergency start of CICS, the MQCONN resource definition must have been installed by the end of the previous CICS run.

5. If you are using the CICS-WebSphere MQ adapter in a CICS system that has interregion communication (IRC) to remote CICS systems, ensure that the IRC facility is OPEN before you start the adapter, by specifying the CICS system initialization parameter IRCSTRT=YES. The IRC facility must be OPEN if the IRC access method is defined as cross-memory; that is, ACCESSMETHOD(XM).
6. 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.
7. Update your CICS CSD as follows:
   a. If you do not share your CSD with earlier releases of CICS, remove the groups CSQCAT1 and CSQCKB, and any copies of those groups or of items from those groups, from your CSD. You must also delete the CKQQ TDQUEUE from group CSQCAT1. The definition for CKQQ is now supplied in the CICS CSD group DFHDCTG.
   b. If you do share your CSD with earlier CICS releases, ensure that CSQCAT1 and CSQCKB, and any copies of those groups or of their content, are not installed for CICS TS 4.1 or CICS TS 3.2. You must also delete the CKQQ TDQUEUE from group CSQCAT1. The definition for CKQQ is now supplied in the CICS CSD group DFHDCTG. For CICS TS releases earlier than CICS TS 3.2, install the CSQCAT1 and CSQCKB groups as part of a group list, after installing DFHLIST, to override group DFHMQ and correctly install the required definitions.
8. Update the WebSphere MQ definitions for the dead-letter queue, default transmission queue, and CICS-WebSphere MQ adapter objects. You can use the sample CSQ4INYG, but you might need to change the initiation queue name to match the default initiation queue name in the MQINI resource definition for your CICS region. You can use this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. Using the CSQUTIL utility is preferable because you do not then have to redefine these objects each time that you restart WebSphere MQ.

The WebSphere MQ transport

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

As a **service provider**, CICS uses WebSphere MQ 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 following information:

- The criteria for when a trigger message is 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, specify that triggering occurs when the first message arrives on the local queue.

  The local queue definition can also specify that trigger data is 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 is processed. In the case of CICS SOAP processing, 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 Task initiator or trigger monitor (CKTI) in Product overview.

You can configure CICS, so that when a message arrives on a local queue, the trigger monitor (provided by WebSphere MQ) 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 is returned on a particular reply queue.

In both cases, CICS and WebSphere MQ require configuration to define the required resources and queues.

Defining local queues in a service provider

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

Procedure

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 for the INITQNAME attribute of the MQINI resource definition for the CICS region. MQINI is an implicit resource that CICS creates when you install an MQCONN resource.
2. For each local request 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; for example, the initiation queue specified in the MQINI definition for the CICS region.
- *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'. The first character must be '/' .
- *nnn* is the number of retries that are attempted.
- *requeuename* is the name of the queue to which failed messages are 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 request queues.

Working with initiation queues

You can inquire on the name of the initiation queue with these interfaces:

CICS Explorer

The CICS Explorer operations views

Use the **Name** attribute in the Websphere MQ Initiation Queues view.

CICSPlex SM

The MQINI operations view

CEMT

The INQUIRE MQINI command

The CICS SPI

The INQUIRE MQINI command

Defining local queues in a service requester

When you use the WebSphere 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.

About this task

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

Procedure

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:

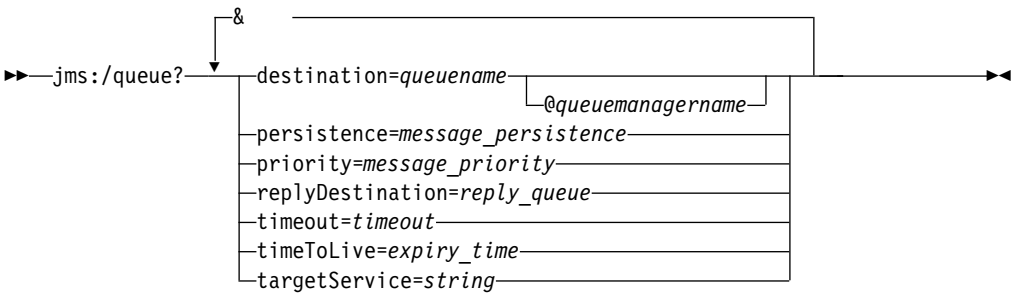
reply_queue is the local queue name.

nnn is the number of retries that will be attempted.

The URI for the WebSphere MQ transport

When communication between the service requester and service provider uses WebSphere 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 WebSphere MQ.

Syntax



CICS uses the following options; other web service providers might use further options that are not described here. The entire URI is passed to the service provider, but CICS ignores any options that it does not support and that are coded in the URI. 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=*queuename* [@*queuemanagername*]

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

queuemanagername is the name of the target queue manager

persistence=*message_persistence*

Specify one of the following options:

0 Persistence is defined by the default queue persistence.

1 Messages are not persistent.

2 Messages are persistent.

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

priority=*message_priority*
Specifies the message priority. CICS supports integer values in the range 0 - 9 for message priorities, where 9 is assigned to the highest priority messages and 0 is assigned to the lowest priority messages. Alternatively, specify **-1** to use the default priority that is defined for the target queue.

replyDestination=*reply_queue*
Specifies the queue to be used for the response message. If this option is not specified, CICS uses 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 waits 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 uses this as the path when attempting to match with URIMAP. If not specified, the value that is specified in TRIGDATA on the input queue at the service provider is used.

This example shows a URI for the WebSphere MQ transport:

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

For information about `"connectionFactory"` and `"initialContextFactory"`, see the *WebSphere MQ product documentation*.

Configuring CICS to support persistent messages

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

About this task

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:

Procedure

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 201. 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.
4. Ensure that during the processing of a persistent message, a program issues an **EXEC CICS SYNCPOINT** command before the first implicit syncpoint is requested; for example, using an SPI command such as **EXEC CICS CREATE TDQUEUE** implicitly takes a syncpoint. Issuing an **EXEC CICS SYNCPOINT** command confirms that the persistent message has been processed successfully. If a program does not explicitly request a syncpoint before trying to implicitly take a syncpoint, an ASP7 abend is issued.

Results

What to do next

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" for details.

Persistent message processing

When a web service request is received in a WebSphere MQ 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 ends abnormally 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 canceled, 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 WebSphere MQ, which might contain RFH2 headers.

   The DFHMQMSG data-container contains the WebSphere MQ 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 WebSphere MQ MQMD control block relating to the **MQ GET** for the original message.

Interoperability between the web services assistant and WSRR

The CICS web services assistant can interoperate with the IBM WebSphere Service Registry and Repository (WSRR). Use WSRR to find web services that you are requesting more quickly and enforce version control of the web services that you are providing.

Both DFHLS2WS and DFHWS2LS include parameters to interoperate with WSRR. DFHLS2WS also includes an optional parameter so that you can add your own customized metadata to the WSDL document in WSRR.

If you want the web services assistant to communicate securely with WSRR, you can use secure socket level (SSL) encryption. Both DFHLS2WS and DFHWS2LS include parameters for using SSL encryption.

To use SSL with the web services assistant and WSRR, see "Example of how to use SSL with the web services assistant and WSRR."

Example of how to use SSL with the web services assistant and WSRR

You can interoperate securely between the web services assistant and an IBM WebSphere Service Registry and Repository (WSRR) server by using secure socket layer (SSL) encryption. To use SSL encryption you need a key store and a trust store; you must also specify certain parameters on the web services assistant.

About this task

Complete the following steps to use SSL encryption for interactions between the web services assistant and WSRR.

Procedure

1. Create a key store for your private keys and public key certificates (PKC).
   a. You can create a key store using a key configuration program such as the IBM Key Management Utility (iKeyman).
   b. Specify the **SSL-KEYSTORE** parameter in DFHWS2LS or DFHLS2WS with the fully qualified name of the key store that you have created.

c. Optional: Specify the **SSL-KEYPWD** parameter in DFHWS2LS or DFHLS2WS with the password of the key store that you have created.

2. Create a trust store for all your trusted root certificate authority (CA) certificates. These certificates are used to establish the trust of any inbound public key certificates.
   a. You can create a trust store using a key configuration program such as the IBM Key Management Utility (iKeyman).
   b. Specify the **SSL-TRUSTSTORE** parameter in DFHWS2LS or DFHLS2WS with the fully qualified name of the trust store that you have created.
   c. Optional: Specify the **SSL-TRUSTPWD** parameter in DFHWS2LS or DFHLS2WS with the password of the trust store that you have created.
3. Test that the web services assistant is able to communicate with WSRR using SSL encryption.
   a. You can use the sample files provided by IBM WebSphere Application Server to test the web services assistant with WSRR.
      - The sample key stores provided by WebSphere Application Server are `DummyClientKeyFile.jks` and `DummyServerKeyFile.jks`.
      - The sample trust stores provided by WebSphere Application Server are `DummyClientTrustFile.jks` and `DummyServerTrustFile.jks`.
   b. Replace the keys in the sample key and trust store files. These keys are shipped with WebSphere Application Server and must be replaced for security.

Results

The web services assistant can now use SSL encryption to communicate securely with WSRR across a network.

The web services infrastructure

CICS applications in a CICS region can either provide a service to, or request a service from, applications that are external to that region by using a web services pipeline. When CICS is a service provider, the CICS application supplies a service to the external application. When CICS is a service requester, the external application supplies a service to the CICS application. Web services pipelines can be configured to use zEnterprise Application Assist Processor (zAAP) where available.

CICS as a service provider

For CICS to provide a service to an external service requester, it must receive the service request and pass it through a pipeline to the target application program. The response from the application is returned to the service requester through the same pipeline.

Figure 21 on page 208 shows an example configuration of the architecture and resources that are required to process a request from an external service requester when CICS is a service provider using a Java pipeline.

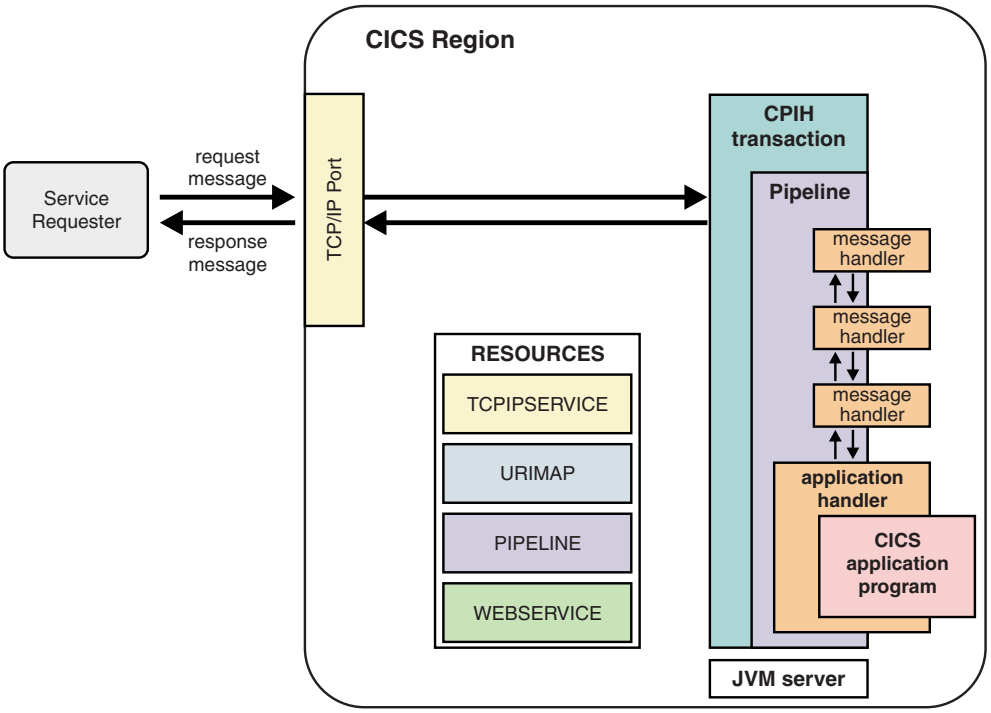


Figure 21. The architecture and resources for a service provider

To process a request, CICS must perform the following operations:

1. Receive the request from the service requester.

   The TCPIPSERVICE resource specifies a port for incoming requests. This port is monitored by the CICS-supplied sockets listener transaction (CSOL).

2. Examine the request, and extract the contents that are relevant to the target application program.

   When the request message is received on the appropriate port, the URIMAP resource definitions are scanned for a URIMAP definition that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the request. If an appropriate URIMAP definition is found, the PIPELINE and WEBSERVICE definitions from the PIPELINE and WEBSERVICE attributes of the URIMAP definition are used. The TRANSACTION attribute of the URIMAP definition determines the name of the transaction that should be attached to process the pipeline. By default the CPIH transaction is used. The URIMAP definition also identifies the PIPELINE and WEBSERVICE resources to use. These resources control the processing that CICS performs.

3. Invoke the application program, passing data extracted from the request.

   The message handlers in the pipeline and the application handler convert the request message into application language structure that the service provider application program expects. The program processes this input and returns a response to the application handler.

4. Construct a response using data returned by the application program, and send a response to the service requester.

   The application handler and message handlers convert the response message received from the service provider application into a message in the format of the original request. This message is sent back to the service requester.

Some of the processing within the pipeline can be performed using the zEnterprise Application Assist Processor (zAAP) if the pipeline is configured appropriately. For more information, see “Java-based SOAP pipelines” on page 210.

CICS as a service requester

For CICS to invoke an external service, an application program sends a request that is passed through a pipeline to a target service. The response from the service is returned to the application program through the same pipeline.

Figure 22 shows an example configuration of the architecture and resources that are required to process a request from a CICS application program for data from a service provider that is external to the CICS region, using a Java pipeline.

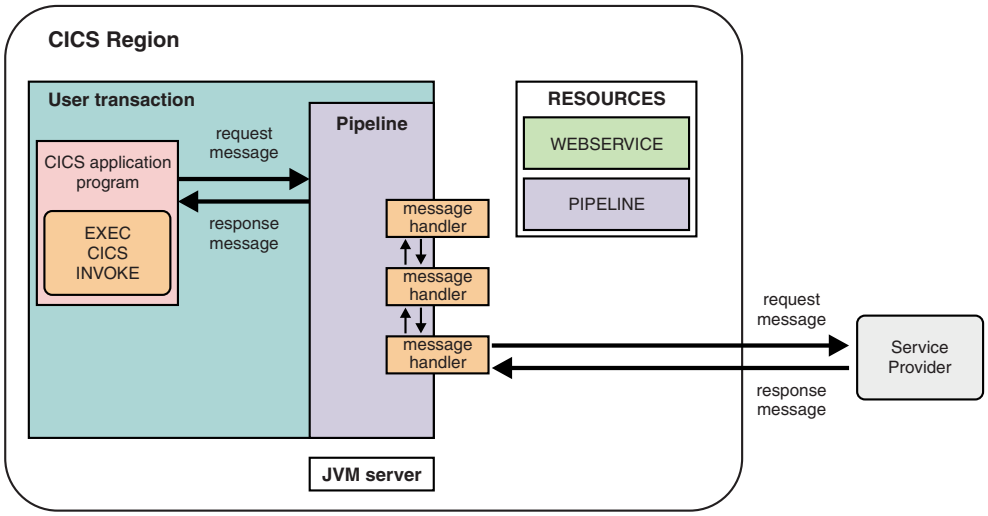


Figure 22. The architecture and resources for a service requester

To process a request, CICS must perform the following operations:

1. Build a request using data provided by the application program.

   When the CICS application program initiates a request to a service provider that is external to the CICS region, the requestor application calls the EXEC CICS INVOKE SERVICE command. The EXEC CICS INVOKE SERVICE command invokes the pipeline. The pipeline converts the application language structure into a language that the service provider can process, for example a SOAP message.
2. Send the request to the service provider.

   CICS sends the request message to the remote service provider by using either HTTP or WebSphere MQ.
3. Receive a response from the service provider.

   When the service provider response message is received, CICS passes the message back to the pipeline.
4. Examine the response, and extract the contents that are relevant to the original application program.

   The pipeline converts the service provider response message into the application language structure, which is passed to the application program. Control is then returned to the application program.

Some of the processing within the pipeline can be performed using the zEnterprise Application Assist Processor (zAAP) if the pipeline is configured appropriately. For more information, see "Java-based SOAP pipelines."

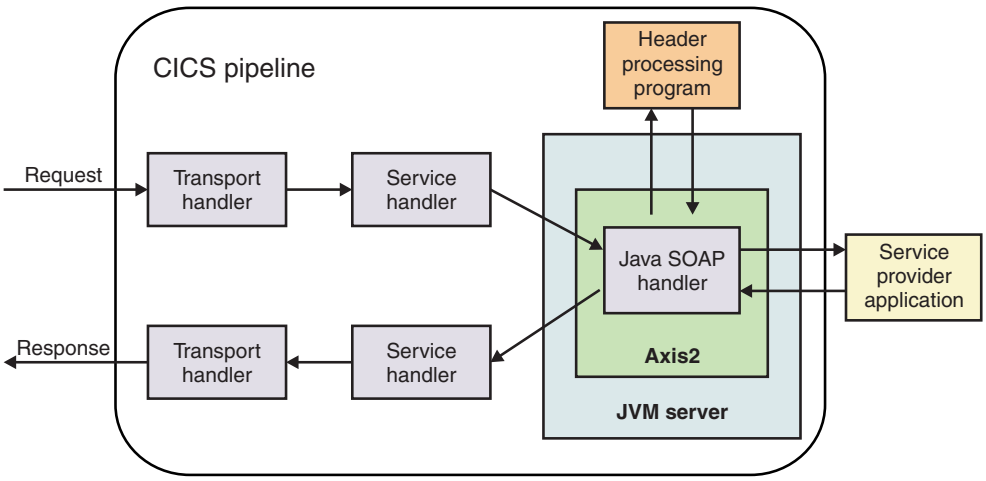
Java-based SOAP pipelines

CICS supports using the Axis2 Java-based SOAP engine to process web service requests in provider and requester pipelines. Because Axis2 uses Java, the SOAP processing is eligible for offloading to the zEnterprise Application Assist Processor (zAAP).

Axis2 is an open source web services engine from the Apache foundation and is provided with CICS to process SOAP messages in a Java environment. You can opt to use Axis2 by adding a Java SOAP handler to your pipeline configuration file and creating a JVM server to handle the Axis2 processing.

Enabling Axis2 does not require regenerating the binding files for any existing web services that use the pipeline. The response times might be slower when using Axis2, but you can offload the SOAP processing to zAAP. For more information about offloading to zAAP, see Java support in CICS in Getting started.

When CICS is a service provider, the Java-based terminal handler uses Axis2 to parse the SOAP envelope for a request message. You can use header processing programs to process any SOAP headers associated with the SOAP message. Axis2 also constructs the SOAP response message. This process is shown in the following diagram:



When CICS is a service requester, the Java-based initial handler in the pipeline uses Axis2 to generate the SOAP envelope for a request message. You can use header processing programs to process any SOAP headers associated with the SOAP message. Axis2 also parses the SOAP response message.

Web service applications and Java

For provider-mode SOAP pipelines, request and response messages are passed between the terminal handler of the pipeline and the web service application by using an application handler. The application handler processes the body of a SOAP request so that the request can be used by the application. The application

handler also generates a response by using the returned data from the application. If the terminal handler of your pipeline is a Java-based message handler, you can specify the supplied Axis2 application handler in the pipeline configuration file, as opposed to specifying the supplied DFHPITP application handler. The application handler processing can then be offloaded to zAAP. For more information about application handlers, see "Application handlers" on page 275.

For requester-mode SOAP pipelines, the web service application invokes the pipeline by using the **EXEC CICS INVOKE SERVICE** command. The request and response messages are then passed between the web service application and the initial handler in the pipeline. If you specify a Java-based handler as the initial handler in the pipeline, then the **EXEC CICS INVOKE SERVICE** command is processed by Axis2, making it possible to offload this process to zAAP. If the first handler is not a Java-based handler, then the **EXEC CICS INVOKE SERVICE** command is processed by CICS.

Axis2 processing in a JVM server

Axis2 requires a JVM server, which is represented by a JVMSERVER resource in CICS. The JVM server is a runtime environment that can handle multiple concurrent requests from different Java programs in a single JVM. The class path for the JVM server must include the Axis2 Java archive files. You can automatically add all of the required JAR files to the class path by specifying the `JAVA_PIPELINE` option in the JVM profile. The pipeline configuration file must also point to the JVMSERVER resource that is configured to support Axis2.

For more information about JVM servers, see Java support in CICS in Getting started.

Axis2 header handlers

Although you can use existing header processing programs, it is more efficient to write Axis2 handlers in Java to process the SOAP headers. These handlers can also run in the JVM server and are therefore eligible for offloading. For more information about creating Axis2 handlers, see Writing Your Own Axis2 Module.

A header handler program can use Axis2 APIs to modify or interact with the Axis2 environment, SOAP messages, and individual web services. Do not use these APIs to customize Axis2, as you might change Axis2 in a way that means CICS cannot run the engine correctly. Axis2 handlers are supported only if they interact with the Axis2 environment in a way that is compatible with how CICS uses Axis2.

Axis2 repository

Axis2 uses a repository to store all of its configuration files, services, and modules. CICS provides a default repository in the *usshome*`/lib/pipeline/repository` directory on z/OS UNIX, where *usshome* is the value of the **USSHOME** system initialization parameter.

The default repository contains the configuration file, `axis2.xml`, which is required by CICS to use Axis2. This file is in the `/conf` subdirectory in the repository. If you create your own repository, you must copy this file to your repository for CICS to work with Axis2.

Do not edit the `axis2.xml` file, unless you are registering handler programs. This file is managed as an internal part of CICS, so you must not make any other

changes to this file unless to do so by IBM support.

Data formatting for Web Services

Different CICS technologies can generate JSON and XML data that is equally specification compliant, but physically different. They might also report errors that are found in an input message in different ways, as a result of the order in which they apply checks to validate the data.

CICS uses several different technologies for automatically transforming JSON and XML data. These include z/OS Connect for CICS (both the Java and non-Java variants), Axis2, and PIPELINE resources. These technologies generate JSON and XML data in an external format as dictated by the relevant specifications.

There can be multiple ways to represent data, that are equally specification-compliant, but physically different. The CICS technologies always generate data that is compliant, but there might be physical differences between them. For example, if you switch between the Java and non-Java z/OS Connect for CICS options for JSON, you might detect minor differences in the generated JSON.

Such differences might include different error messages being reported under failure conditions, differences in how white-space characters are inserted, alternative (but equivalent) representations for numeric data, and variations in how special characters are escaped. Further changes of this nature can be introduced as a result of applying corrective maintenance to CICS, or with new releases of CICS.

Partner systems, such as a JSON client, should be written to tolerate specification-compliant variations of this nature. Partners often exploit widely used cross-industry libraries and technologies for interacting with JSON and XML; such libraries automatically handle such minor formatting differences. However, it is possible for less-compliant partner systems to detect and respond differently to the formatting differences in the various CICS technologies, care might therefore be required if you are writing applications that work directly with the JSON or XML without the benefit of a standards-based parser.

CICS as a service provider for JSON requests

For CICS to provide a service to an external JSON client, it must receive the request and pass it through a pipeline to the target application program. The response from the application is returned to the JSON client through the same pipeline.

There are three ways of configuring CICS as a service provider for JSON requests:

- Using z/OS Connect for CICS.

  z/OS Connect for CICS is a technology for accessing z/OS assets, such as CICS programs, by using JSON. For more information about z/OS Connect for z/OS, and some of the differences between z/OS Connect for CICS and CICS Java pipelines, see .
- Using CICS Java pipelines.

  CICS Java pipelines are the technology that is provided in previous versions of CICS to allow access to CICS programs by using JSON. For more information about CICS Java pipelines, see Java-based SOAP pipelines in Configuring.
- Using the JAX-RS and JSON features of the CICS Liberty JVM server directly.

CICS receives JSON data and transforms it into structured application data that is understood by the CICS application program. The responses from the CICS application are transformed into a JSON payload for the outbound response. The transformations require parsing of the messages. If you use z/OS Connect for CICS, you have the option of using a Java-based JSON parser or a non-Java equivalent. The configuration option is specified in the z/OS Connect for CICS pipeline configuration file. For more information about configuring z/OS Connect for CICS, see . If you use CICS Java pipelines, the parsing is only ever performed by using Java within the JVM server. The implications of the different configuration decisions are:

- If you parse JSON by using Java, the processing is eligible for offloading to a zEnterprise Application Assist Processor (zAAP) if it is available. Offloading the processing might have cost benefits.
- If you use the z/OS Connect for CICS non-Java parser, some workloads might get performance and throughput benefits. For more information about which workloads might benefit, see the performance material that is made available with this release. Even if you use the non-Java parser, most of the z/OS Connect for CICS infrastructure processing is eligible for offloading onto a zEnterprise Application Assist Processor (zAAP).

CICS as a service provider for JSON requests using z/OS Connect for CICS

For CICS to provide a service to an external JSON client, it must receive the request into a z/OS Connect for CICS, transform the JSON message, and pass it to the target application program. The response from the application is returned to the JSON client through the same mechanism.

Figure 23 on page 214 shows an example configuration of the architecture and resources that are required to process a request from an external JSON client when CICS is a service provider that uses z/OS Connect for CICS.

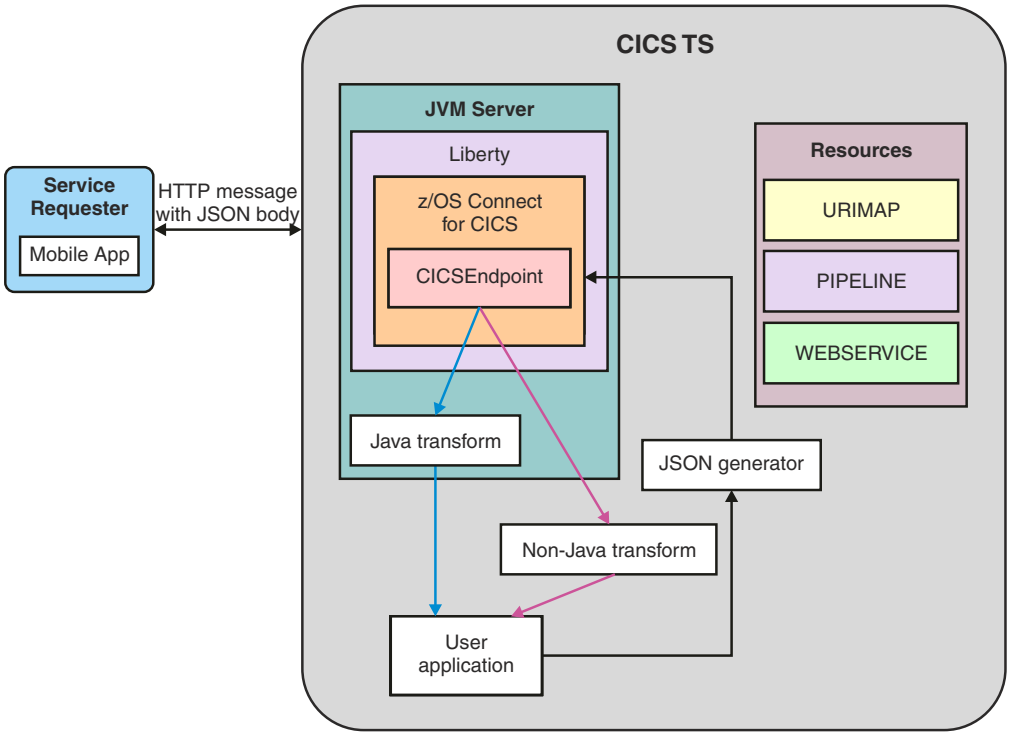


Figure 23. The architecture and resources for a JSON service provider that uses z/OS Connect for CICS

Processing a JSON request with z/OS Connect for CICS

To process a request, CICS completes the following operations:

1. Receive the request from the service requester.

   WebSphere Liberty receives the request and passes it to z/OS Connect for CICS.

2. CICS resolves the URIMAP resource for the request, by scanning the URIMAP definitions with a USAGE attribute set to JVMSERVER. The URIMAP identifies the WEBSERVICE resource that is used. A new CICS task is started to process the request by using the transaction ID from the URIMAP. By default, the CPIH transaction is used.

   The WEBSERVICE resource controls the processing that CICS performs. In particular, the WSBind file pointed to by the WEBSERVICE resource is used for data transformation between JSON and structured application data. WSBind files for JSON web services are generated by using utilities DFHLS2JS and DFHJS2LS.

   Note: Runtime validation of JSON data against schema is not supported. The value of the VALIDATION attribute of a WEBSERVICE resource that is used with a JSON payload is ignored.
   For information about any restrictions that apply, see JSON web service restrictions.

3. z/OS Connect for CICS processes the request according to how it is configured. If z/OS Connect for CICS is configured to use global interceptors, the interceptors run during this processing.

4. The CICSEndpoint receives control. The JSON payload is transformed to structured application data according to the configuration option specified in the pipeline configuration file. There are two options for how the transformation is performed:
   - Java transformation is performed within the JVM server. This processing is eligible for offload to a zAAP processor if one is available.
   - Non-Java transformation is performed outside of the JVM server and might provide performance and throughput benefits for certain workloads. For more information about which workloads might benefit, see the performance material that is made available with this release.

   The mapping is performed according to the information in the WSBind file. The output from the transform is equivalent in both cases.
5. CICS links to the application program and passes the transformed data. The program processes this input and returns a response to the JSON generator.
6. The JSON generator generates a JSON response message by using the output from step 5. This message is sent back to the service requester via z/OS Connect for CICS.

z/OS Connect for CICS also provides a RESTful interface that supports the standard RESTful methods:

POST

PUT

GET

DELETE

HEAD

Where appropriate, this interface uses the transform and generator that the pipeline is configured for, in the same way as do normal JSON requests. For more information about RESTful JSON web services, see Concepts of RESTful JSON web services in Getting started.

Most of the z/OS Connect for CICS infrastructure processing is eligible for offloading onto a zEnterprise Application Assist Processor (zAAP).

CICS as a service provider for JSON requests using CICS Java pipelines

For CICS to provide a service to an external JSON client, it must receive the request and pass it through a pipeline to the target application program. The response from the application is returned to the JSON client through the same pipeline. The JSON transform is performed by using Java within the JVM server.

Figure 24 on page 216 shows an example configuration of the architecture and resources that are required to process a request from an external JSON client when CICS is a service provider that uses a Java pipeline. The pipeline processing for a JSON request is similar to the way that CICS processes a SOAP request in a Java pipeline. For more information, see Java-based SOAP pipelines in Configuring.

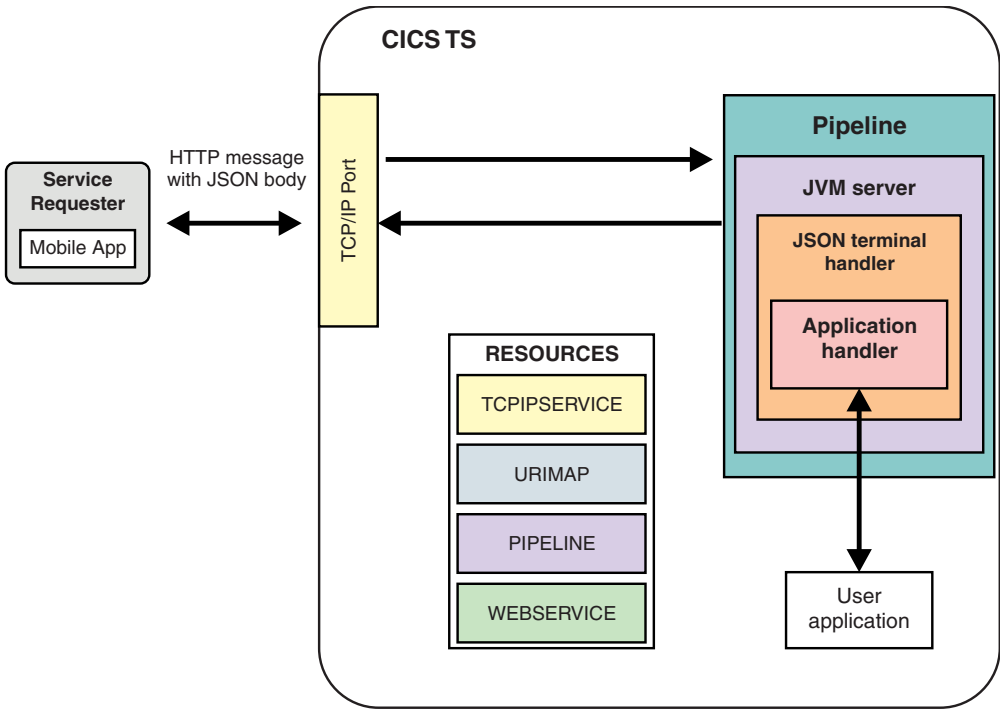


Figure 24. The architecture and resources for a JSON service provider that uses a CICS Java pipeline

Processing a JSON request

To process a request, CICS completes the following operations:

1. Receive the request from the service requester.

   The TCPIPSERVICE resource specifies a port for incoming requests. This port is monitored by the CICS-supplied sockets listener task (CSOL).

2. Examine the request, and extract the contents that are relevant to the target application program.

   When the request message is received on the appropriate port, the URIMAP resource definitions are scanned for a URIMAP definition that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the request. If an appropriate URIMAP definition is found, the PIPELINE and WEBSERVICE definitions from the PIPELINE and WEBSERVICE attributes of the URIMAP definition are used. The TRANSACTION attribute of the URIMAP definition determines the name of the transaction that should be attached to process the pipeline. By default the CPIH transaction is used. The URIMAP definition also identifies the PIPELINE and WEBSERVICE resources to use.

   These PIPELINE and WEBSERVICE resources control the processing that CICS performs. In particular, the WSBind file pointed to by the WEBSERVICE resource is used for data transformation between JSON and language structures. WSBind files for JSON web services are generated by using utilities DFHLS2JS and DFHJS2LS.

   Note: Runtime validation of JSON data against schema is not supported. The value of the VALIDATION attribute of a WEBSERVICE resource that is used with a JSON payload is ignored.
   For information about any restrictions that apply, see JSON web service restrictions.

3. Pipeline processing begins and the request flows through any handlers that are defined. It is not expected that any of the handlers that are currently provided by CICS for SOAP web services will be relevant to JSON web services.
4. At the end of the pipeline, the JSON terminal handler is called. This terminal handler is a Java program that interfaces with the Axis2 pipeline. The terminal handler performs the necessary setup of the Axis2 configuration and then starts the Axis2 engine with the HTTP request body. Within the Axis2 pipeline, the JSON body (if present) is parsed and a Java object model that represents the contents is constructed. CICS then calls the application handler. The main role of the application handler is to map the Java object model representation of the request into application data. This mapping is performed by using the description of the language structure in the WSBind file.
5. Call the application program, passing data that is extracted from the request.

   Then the application handler links to the application program. The program processes this input and returns a response to the application handler.
6. Construct a response by using data returned by the application program, and send a response to the service requester.

   The application handler and message handlers convert the response message received from the service provider application into a message in the format of the original request. This message is sent back to the service requester.

Some of the processing within the pipeline is eligible for offloading onto a zEnterprise Application Assist Processor (zAAP).

Creating the CICS infrastructure for a SOAP service provider

To create the CICS infrastructure for a SOAP service provider, you must create a pipeline configuration file and create a number of CICS resources.

Before you begin

If you want to use a Java pipeline, ensure that a JVMSERVER resource exists with the JAVA_PIPELINE=YES option specified in the JVM Profile.

A JVM server can handle SOAP processing for many Java pipelines.

About this task

You can define the PIPELINE resource in a local CICS region using CICS or CICSPlex SM functions, or you can use the CICS Explorer to define the PIPELINE resource either in a local CICS region or in a CICS bundle. When you use the CICS Explorer to define a PIPELINE resource in a CICS bundle, you also create the pipeline configuration file and package it in the CICS bundle, so you do not have to manage this file separately. PROGRAM resources and WEBSERVICE resources can also be defined in CICS bundles. When you define a WEBSERVICE resource in a CICS bundle, you can import a web service binding file and a WSDL document or WSDL archive file and include these in the bundle. You can also create URIMAP definitions to support the web service and package these in a bundle. For more help with using the CICS Explorer to create and edit resources in CICS bundles, see Working with bundles in the CICS Explorer product documentation.

Procedure

1. Define the transport infrastructure.

a. If you are using the WebSphere 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.
   1) See Configuring CICS to use the WebSphere MQ transport in Configuring for details.
b. If you are using the HTTP transport, you must define a TCPIPSERVICE resource that defines the port on which inbound requests are received.
   1) See CICS resources for web services in Configuring for details.

2. Optional: Repeat this step for each different transport configuration you need.
3. Define the message handlers and header processing programs that you want to include in the pipeline configuration file to process inbound web service requests, and their responses. CICS provides the following handlers and header processing programs:
   a. SOAP message handlers, to process SOAP 1.1 or 1.2 messages. You can support only one level of SOAP in a service provider pipeline.
   b. MTOM handler, to process MIME Multipart/Related messages that conform to the MTOM/XOP specifications.
   c. Support for securing web services in Securing, to process secure web service messages.
   d. Support for Web Services Transactions in Configuring, to process atomic transaction messages.
4. Optional: If you want to perform your own processing in the pipeline, you must create a message handler or header processing program. See Message handlers in Configuring for details. If you decide to create custom message handler programs, to optimize performance you must make them threadsafe.
5. Create an XML pipeline configuration file containing your message handlers, header processing programs, and application handler.
   a. CICS provides two basic provider mode pipeline configuration file samples, `basicsoap11provider.xml` and `basicsoap11javaprovider.xml`.
   b. You can edit these samples, or add additional message handlers as appropriate. The samples are provided in the library `/usr/lpp/cicsts/cicsts53/samples/pipelines` (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX).
   c. For more information about options available in the pipeline configuration file, see Pipeline configuration files in Configuring
6. Copy the pipeline configuration file to a suitable directory in z/OS UNIX.
7. Change the pipeline configuration file permissions to allow the CICS region to read the file.
8. Repeat steps 5 through 7 for each different pipeline configuration that you require.
9. Create a PIPELINE resource.
   a. The PIPELINE resource defines the location of the pipeline configuration file. It also specifies a *pickup directory*, which is the z/OS UNIX directory that contains the web service binding files and optionally the WSDL.
   b. Repeat this step for each different pipeline configuration.
   a. When you create a PIPELINE resource, CICS reads any files in the specified pickup directory, and creates the WEBSERVICE resource and URIMAP resource dynamically.

10. Unless you use autoinstalled PROGRAM definitions, create a PROGRAM resource for each program that runs in the pipeline. These include the target application program, which normally runs 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.

Results

Your CICS system now contains the infrastructure needed for each service provider.

What to do next

You can extend the configuration when you need to do so, either to define additional transport infrastructure, or to create additional pipelines.

Creating the CICS infrastructure for a SOAP service requester

To create the CICS infrastructure for a SOAP service requester, you must create a pipeline configuration file and create a number of CICS resources.

Before you begin

If you want to use a Java pipeline, ensure that a JVMSERVER resource exists with the JAVA_PIPELINE=YES option specified in the JVM Profile. See JVMSERVER resources in Reference -> System definition.

A JVM server can handle SOAP processing for many Java pipelines.

About this task

You can define the PIPELINE resource in a local CICS region using CICS or CICSPlex SM functions, or you can use the CICS Explorer to define the PIPELINE resource either in a local CICS region or in a CICS bundle. When you use the CICS Explorer to define a PIPELINE resource in a CICS bundle, you also create the pipeline configuration file and package it in the CICS bundle, so you do not have to manage this file separately. PROGRAM, WEBSERVICE and URIMAP resources can also be defined in CICS bundles. When you define a WEBSERVICE resource in a CICS bundle, you can import a web service binding file and a WSDL document or WSDL archive file and package these in the bundle, and for a service provider you can choose to include a PROGRAM definition. You can also create URIMAP definitions to support the web service and package these in a bundle. For more help with using the CICS Explorer to create and edit resources in CICS bundles, seeWorking with bundles in the CICS Explorer product documentation.

Procedure

1. Define the message handlers and header processing programs that you want to include in the pipeline configuration file to process inbound web service requests, and their responses. CICS provides the following handlers and header processing programs:
   a. SOAP message handlers, to process SOAP 1.1 or 1.2 messages. You can only support one level of SOAP in a service requester pipeline.
   b. MTOM handler, to process MIME Multipart/Related messages that conform to the MTOM/XOP specifications.
   c. Security handler, to process secure web service messages.

d. WS-AT header processing program, to process atomic transaction messages.

2. Optional: If you want to perform your own processing in the pipeline, you must create a message handler or header processing program. See "Message handlers" on page 277 for details. If you decide to create custom message handler programs, to optimize performance you must make them threadsafe.
3. Create an XML pipeline configuration file containing your message handlers and header processing programs. CICS provides two basic requester mode pipeline configuration file samples, `basicsoap11provider.xml` and `basicsoap11javaprovider.xml`, which you can copy and edit as appropriate. These samples are provided in the library `/usr/lpp/cicsts/cicsts53/samples/pipelines` (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX). For more information about options available in the pipeline configuration file, see "Pipeline configuration files" on page 226
4. Copy the pipeline configuration file to a suitable directory in z/OS UNIX.
5. Change the pipeline configuration file permissions to allow the CICS region to read the file.
6. Repeat steps 3 to 5 for each different pipeline configuration that you require.
7. Create a PIPELINE resource. See PIPELINE resources in Reference -> System definition. The PIPELINE resource defines the location of the pipeline configuration file. It also specifies a *pickup directory*, which is the z/OS UNIX directory that contains the web service binding files and optionally the WSDL. You can also specify a timeout in seconds, which determines how long CICS waits for a response from web service providers. Repeat this step for each pipeline configuration file. When you create a PIPELINE resource, CICS reads any files in the specified pickup directory and creates the WEBSERVICE resources dynamically (see WEBSERVICE resources in Reference -> System definition).
8. Unless you use autoinstall PROGRAM definitions, create a PROGRAM resource for each program that runs in the pipeline. See PROGRAM resources in Reference -> System definition. These programs include the service requester application program, which normally runs 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.
9. Optional: Create a URIMAP resource (see URIMAP resources in Reference -> System definition) for client requests to each URI that your service requesters use to make requests, following the instructions in Creating a URIMAP resource for CICS as a HTTP client in Developing applications. You can specify the URI directly on the **INVOKE SERVICE** command in your programs, instead of using a URIMAP resource. However, using a URIMAP resource means that you do not need to recompile your applications if the URI of a service provider changes. With a URIMAP resource you can also choose to implement connection pooling, where CICS keeps the client connection open after use, so that it can be reused by the application for subsequent requests, or by another application that calls the same service.

Results

Your CICS system now contains the infrastructure needed for each service requester.

What to do next

You can extend the configuration when you need to do so, to create additional pipelines.

Creating the CICS infrastructure for a JSON service provider

To create the CICS infrastructure for a JSON service provider, you must create a pipeline configuration file and create a number of CICS resources.

Before you begin

To use CICS as a service provider for JSON requests or use the linkable interface to transform JSON, define and install a JVMSERVER resource with a JVM profile that has the **JAVA_PIPELINE=YES** option specified. An example JVMSERVER resource definition called DFHAXIS is provided in group DFH$AXIS.

Note: The infrastructure described here assumes that you are not using z/OS Connect for CICS to connect to your JSON service provider, and consequently it uses Java parsing within the JVM server to parse the JSON messages. If you want to use non-Java JSON parsing, you must use z/OS Connect for CICS to connect to the JSON web service. For more information about setting up z/OS Connect for CICS, see .

Procedure

1. Define the transport infrastructure.
   - Define a TCPIPSERVICE resource that defines the port on which inbound requests are received. See CICS resources for web services for details.
2. Define the message handlers that you want to include in the pipeline configuration file to process inbound web service requests, and their responses.

   If you want to perform your own processing in the pipeline, you must create a message handler. See Message handlers for details. If you decide to create custom message handler programs, to optimize performance you must make them threadsafe.
3. Create an XML pipeline configuration file containing your message handlers, header processing programs, and application handler. CICS provides a basic provider mode pipeline configuration file sample, `jsonjavaprovider.xml`. You can edit this sample to add additional message handlers as appropriate. This sample is provided in the directory `/usr/lpp/cicsts/cicsts53/samples/pipelines`, where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX). For more information about options available in the pipeline configuration file, see “Elements used in service provider and service requester pipelines” on page 241.
4. Copy the pipeline configuration file to a suitable directory in z/OS UNIX.
5. Change the pipeline configuration file permissions to allow the CICS region to read the file.
6. Create a PIPELINE resource. The PIPELINE resource defines the location of the pipeline configuration file. It also specifies a *pickup directory*, which is the z/OS UNIX directory that contains the web service binding files. Repeat this step for each different pipeline configuration. When you install a PIPELINE resource or perform a PIPELINE SCAN, CICS reads the `.wsbind` files in the specified pickup directory, and creates appropriate WEBSERVICE and URIMAP resources dynamically.
7. Unless you use autoinstalled PROGRAM definitions, create a PROGRAM resource for each program that runs in the pipeline. These include the target application program, which normally runs 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.

Results

You have created the infrastructure needed for each service provider and you can now install these resources on your CICS system.

What to do next

Install the resources. You can extend the configuration when you need to do so, either to define additional transport infrastructure, or to create additional pipelines.

Configuring z/OS Connect for CICS

z/OS Connect for CICS is distributed as part of CICS Transaction Server. Users can configure z/OS Connect for CICS manually, and it is typically performed one time, with further configuration for individual services.

Before you begin

Consider whether you already have a WebSphere Liberty Profile JVM server that is configured in CICS. Although it is possible to host z/OS Connect and other unrelated services in the same WebSphere Liberty Profile environment, it is good practice to configure a separate JVM server for the sole use of z/OS Connect. It is also good practice to have only a single WebSphere Liberty Profile JVM server that is configured in any single CICS region. You can host z/OS Connect in its own CICS region, or group of CICS regions, and use the Distributed Program Link mechanism to call CICS programs in the application-owning CICS regions.

Procedure

1. Create a JVMSERVER configured to support the WebSphere Liberty Profile. For more information about creating a WebSphere Liberty Profile JVMSERVER, see .
2. WebSphere Liberty Profile will require configuration to meet your security requirements. z/OS Connect for CICS requires the use of SSL. By default it expects the use of client-certified SSL certificates; you can add the following configuration option to the `server.xml` file to enable the use of HTTP Basic Authentication:

   ```
   <!-- Allow fail-over to HTTP Basic Authentication -->
   <webAppSecurity allowFailOverToBasicAuth="true"/>
   ```

   For more information about WebSphere Liberty Profile security, see Configuring security for a Liberty JVM server in Securing.
3. Locate the `server.xml` file for the WebSphere Liberty Profile, and update the `<featureManager>` list to include a `<feature>cicsts:zosConnect-1.0</feature>` feature as shown in the following example:

   ```
   <featureManager>
        <feature>cicsts:core-1.0</feature>
        <feature>ssl-1.0</feature>
        <feature>cicsts:zosConnect-1.0</feature>
    </featureManager>
   ```
4. Add the following statement to the `server.xml` file to define the z/OS Connect for CICS receiver program:

   ```
   <com.ibm.cics.wlp.zosconnect.CICSEndpoint
          id="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
   ```
5. Install the JVMSERVER. Review the contents of the generated `messages.log` file. This log contains the messages that are generated by WebSphere Liberty Server, including messages that are issued by z/OS Connect for CICS such as:

```
SRVE0169I: Loading Web Module: z/OS Connect.
SRVE0250I: Web Module z/OS Connect has been bound to default_host.
```

Check the log for any error or warning messages.

6. Create an XML pipeline configuration file. Sample pipeline configuration file `jsonzosconnectprovider.xml` is provided in the directory `/usr/lpp/cicsts/cicsts53/samples/pipelines/` (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX). You must decide whether you want to parse the JSON by using Java in the Liberty JVM server (the default), or to use the non-Java JSON parser. If you want to use the non-Java JSON parser you must include a `java_parser="no"` attribute on the `<provider_pipeline_json>` element.
   - To parse the JSON by using Java in the Liberty JVM server, you can use the sample pipeline configuration file, but replace `DFHWLP` in the `<jvmserver>` element with the name of your JVMSERVER from step 1.
   - To parse the JSON by using the non-Java parser, modify the sample configuration file to append the `java_parser="no"` attribute to the `<provider_pipeline_json>` element as in the following example:

     ```
     <?xml version="1.0" encoding="EBCDIC-CP-US"?>
     <provider_pipeline_json java_parser="no"
       xmlns="http://www.ibm.com/software/htp/cics/pipeline">
        <jvmserver>DFHWLP</jvmserver>
     </provider_pipeline_json>
     ```

   Replace `DFHWLP` with the name of your JVMSERVER from step 1.
7. Copy the pipeline configuration file to a suitable directory in zFS and ensure that the file permissions allow the CICS region to read the file.
8. Create a PIPELINE resource. The PIPELINE resource defines the location of the pipeline configuration file. It also specifies an optional web service binding directory (also known as a *pickup directory*), which is the zFS directory that contains the web service binding `.wsbind` files. When you install a PIPELINE resource or perform a PIPELINE SCAN, CICS reads the `.wsbind` files in the specified pickup directory, and creates appropriate WEBSERVICE and URIMAP resources dynamically.

Results

Your z/OS Connect for CICS instance is configured. You can test the basic configuration for z/OS Connect by typing this URL into a web browser: `https://`*`hostname`*`:`*`portnumber`*`/zosConnect/apim/services`, where *hostname* is the IP address or host name of the system on which the CICS region that is hosting z/OS Connect for CICS is running, and *portnumber* is the **`httpsPort`** that is specified in the `<httpEndpoint>` element of the `server.xml` file. The web browser displays a list of installed services; because no services are yet installed, the list is empty.

What to do next

You are now ready to install JSON Web Services into z/OS Connect for CICS. For more information, see .

Related information:

Configuring z/OS Connect for CICS for a CICS JSON web service

After initially configuring z/OS Connect for CICS, you can configure it for a CICS JSON web service. JSON web services are deployed to z/OS Connect for CICS in a similar way as to other CICS PIPELINE environments.

Before you begin

You must complete "Configuring z/OS Connect for CICS" on page 222 before you start this task. You also require a JSON WSBind file for each service you want to deploy; these can be generated by using the JSON Assistants **DFHLS2JS**and **DFHJS2LS**.

About this task

JSON web services are deployed to z/OS Connect for CICS in a similar fashion to how they are deployed to other CICS PIPELINE environments.

Procedure

- If your application is simple, you can use the PIPELINE scan command to automatically configure a web service for z/OS Connect for CICS. Deploy the JSON WSBind file to the PIPELINE that is configured for z/OS Connect for CICS, and issue the **PIPELINE SCAN** command. If a URI was associated with the WSBind file when it was created, that URI is dynamically registered with z/OS Connect for CICS and no further action is required. If your **WEBSERVICE** is dynamically registered with z/OS Connect for CICS it will use the name of the **WEBSERVICE** resource prefixed with **CICSService_**. For example, if a **WEBSERVICE** called TESTWEBS is scanned in to z/OS Connect for CICS it will be known as CICSService_TESTWEBS. If your application is more complex, extra configuration steps are required; these are listed under the next bullet point.
- If your application is more complex, use the following steps to deploy it:
  1. Place the WSBind file in the PIPELINE's WSDIR and issue a **PIPELINES sCAN** command, or install a **WEBSERVICE** resource that associates the WSBind file with the appropriate **PIPELINE** resource.
  2. Ensure that a **URIMAP** resource is installed for the **WEBSERVICE**. If the **WEBSERVICE** was installed by using a **PIPELINE SCAN**, and indicates the URI at which the service is installed, the **URIMAP** is created automatically. Otherwise, a **URIMAP** can be manually defined. The **URIMAP** must specify **USAGE(JVMSERVER)** and have the **PIPELINE** and **WEBSERVICE** attributes set.
  3. Locate the Liberty JVM server configuration file, `server.xml`. An entry is needed in this file to associate the URI of the Service with the z/OS Connect for CICS receiver. The use of a trailing `*` wildcard can be used to associate many URIs with the endpoint. Use of the `*` wildcard can make this step unnecessary if a suitable naming convention is applied. For example:

     ```
     <zosConnectService invokeURI="/json*" serviceName="CICSService1"
          serviceRef="com.ibm.cics.wlp.zosconnect.CICSEndpointService"/>
     ```

     In this example, all URIs that begin with`"/json"` are associated with the z/OS Connect for CICS receiver program, under the name `CICSService1`. Subsequent services with a URI beginning `/json` do not need extra configuration. You can add extra entries. Each entry has a unique `serviceName` and `invokeURI`, but use the same value for the `serviceRef`. You

do not need to restart the Liberty JVM server for this change to the `server.xml` configuration file to be used.

Results

You can test the configuration for z/OS Connect for CICS by typing this URL into a web browser: `https://hostname:portnumber/zosConnect/apim/services`. The *hostname* is the IP address or name of the CICS region that hosts z/OS Connect for CICS. The *portnumber* is the **`httpsPort`** that is configured in the `<httpEndpoint>` section of the `server.xml` configuration file. The web browser displays a list of the installed services.

The service is now ready to be called from a JSON client that uses the same *hostname* and *portnumber* as explained previously. The rest of the URI is formed as defined in the CICS URIMAP resource. When a request arrives in the Liberty JVM server, it is associated with the z/OS Connect for CICS receiver that uses the information from the `server.xml` configuration file. A new CICS task starts to perform this work, and it is associated with a specific **`WEBSERVICE`** resource that uses the information from the URIMAP resource. The data transformation process occurs in the Liberty JVM server, and the target CICS program is attached, as named in the WSBind file.

Related information:

Moving JSON web services from a Java Pipeline to z/OS Connect for CICS

You can move JSON web services from a Java Pipeline to z/OS Connect for CICS.

Before you begin

You need an existing JSON web service that is deployed to a Java Pipeline. Suitable examples are available in the IBM Redbooks: Implementing IBM CICS JSON Web Services for Mobile Applications Redbooks publication.

About this task

Redeploying a JSON web service from a Java Pipeline (as used with the CICS Transaction Server Feature Pack for Mobile Extensions V1.0) to z/OS Connect for CICS is a straightforward process. The z/OS Connect environment in CICS is compatible with the Java Pipeline technology. Artifacts that are developed and deployed to a Java Pipeline can be redeployed for use with z/OS Connect for CICS. There are JSON examples in the IBM Redbooks: Implementing IBM CICS JSON Web Services for Mobile Applications Redbooks publication, which can be used with z/OS Connect for CICS.

The WSBind files that are used for the Java Pipeline and for z/OS Connect for CICS are produced by using the same tools (DFHLS2JS and DFHJS2LS), and are fully compatible with each other.

The z/OS Connect for CICS environment requires the use of SSL between the client application and CICS. If your existing Java Pipeline environment does not use SSL, the conversion to z/OS Connect for CICS involves an extra step.

Procedure

1. Create the necessary z/OS Connect for CICS infrastructure by using the instructions from . For part of this configuration, you select an SSL TCP/IP port number at which the Liberty JVM server listens for incoming connections. You have two options to consider:
   a. Select a different port number from the port number that is used by the TCPIPSERVICE for the Java Pipeline. This option has the advantage of ensuring that both environments can be installed concurrently on different TCP/IP ports. It means that client programs need updating to target the new JSON Service. If the old environment didn't use SSL, the conversion to z/OS Connect for CICS necessitates changes to the URI, so this option is more suitable.
   b. Select the same port number as that using by the TCPIPSERVICE for the Java Pipeline. The port numbers in the URIs used by the client programs do not need to be changed, but the two environments cannot be installed concurrently. The URI might need changing for other reasons, such as switching from HTTP to HTTPS when you enable SSL.
2. Deploy the WSBind files to z/OS Connect for CICS. Your existing WSBind files are entirely compatible with z/OS Connect for CICS. Follow the steps to deploy a new JSON web service to z/OS Connect for CICS as described in . CICS does not allow two WEBSERVICE resources with the same name to both be installed. You therefore have two options:
   a. Discard the original WEBSERVICE to allow the new one to install with the same name as was previously used.
   b. Rename the new WEBSERVICE to avoid a clash.
3. If you customized the processing of the Java Pipeline through use of PIPELINE Handler programs, consider whether that customization is still needed. If it is needed, create z/OS Connect Interceptor programs with equivalent functions, and deploy them as global interceptors. For more information about z/OS Connect interceptors, see Defining z/OS Connect interceptors in IBM HTTP Server 8.5.5 product documentation.

Results

You are now ready to test your new z/OS Connect for CICS JSON web services. If you deployed the services by using the same port number as used in the Java Pipeline, no changes are needed in the client (unless the security configuration has changed, such as when enabling SSL). If you changed the port number or URI for the service, the client needs changing.

Related information:

Pipeline configuration files

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 file system 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. The XML schemas for the pipeline configuration files are in the directory `/usr/lpp/cicsts/cicsts53/schemas/pipeline/` (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX). When you work with configuration files, ensure that the character set

encoding is UTF-8. If you import an existing configuration file that is encoded in EBCDIC, it is automatically converted to UTF-8.

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 needs several different pipeline configurations. However, where different applications have similar requirements, they can share the same pipeline configuration.

Note: When using CICS Explorer to create a new PIPELINE configuration file as part of a bundle, there must not be a configuration file with the same name in the root of the bundle.

There are two kinds of pipeline configurations: 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 `<provider_pipeline>` and `<requester_pipeline>` elements have the following immediate sub-elements:

- 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 optional `<apphandler>` element, which is used to specify channel-attached application handlers.
- For the `<provider_pipeline>` only, an optional `<apphandler_class>` element, which is used to specify an Axis2 application handler.
- An optional `<service_parameter_list>` element, which contains the parameters that are available to the message handlers in the pipeline.

Certain elements can have attributes associated with them. Each attribute value must have quotes around it to produce a valid XML document.

Associated with the pipeline configuration file is a PIPELINE resource. The attributes include CONFIGFILE, which specifies the name of the pipeline configuration file in z/OS UNIX. 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 configuration files. They are provided in library `/usr/lpp/cicts/cicsts53/samples/pipelines`.

basicsoap11provider.xml
: A service provider pipeline definition that uses the SOAP 1.1 protocol for a pipeline that does not support Java. The pipeline uses the `<cics_soap_1.1_handler>` message handler and is used when the CICS application has been deployed using the CICS web services assistant.

basicsoap11requester.xml
: A service requester pipeline definition that uses the SOAP 1.1 protocol for a pipeline that does not support Java. The pipeline uses the `<cics_soap_1.1_handler>` message handler and is used when the CICS application has been deployed using the CICS web services assistant.

basicsoap11javaprovider.xml
: A service provider pipeline definition that uses the SOAP 1.1 protocol for a pipeline that supports Java. The pipeline uses the `<cics_soap_1.1_handler_java>` message handler and is used when the application has been deployed using the CICS web services assistant. This configuration contains the element `<jvmserver>`. This message handler must be edited to specify the appropriate JVM server before the configuration can be used.

basicsoap11javarequester.xml
: A service requester pipeline definition that uses the SOAP 1.1 protocol for a pipeline that supports Java. The pipeline uses the `<cics_soap_1.1_handler_java>` message handler and is used when the application has been deployed using the CICS web services assistant. This configuration contains the element `<jvmserver>`. This message handler must be edited to specify the appropriate JVM server before the configuration can be used.

jsonjavaprovider.xml
: A service provider pipeline definition that uses the JSON message format for a pipeline that supports Java. The pipeline uses the `<cics_json_handler_java>` message handler and is used when the CICS application has been deployed using the CICS JSON assistant. This configuration contains the element `<jvmserver>`. This message handler must be edited to specify the appropriate JVM server before the configuration can be used.

jsonzosconnectprovider.xml
: A pipeline definition for a JSON web service that is deployed to a PIPELINE that is configured for z/OS Connect for CICS. The pipeline uses the `<provider_pipeline_json>` message handler. This configuration contains the element `<jvmserver>`. This message handler must be edited to specify the appropriate JVM server before the configuration can be used.

kerberosprovider.xml

: A service provider pipeline definition that adds configuration information for Kerberos support to `basicsoap11provider.xml`.

samlprovider.xml

: A service provider pipeline definition that adds configuration information for SAML support to `basicsoap11provider.xml`.

samlrequester.xml

A service requester pipeline definition that adds configuration information for SAML support to basicsoap11requester.xml.

propagatesamlprovider.xml

A service provider pipeline definition that adds configuration information for SAML support with propagation of SAML information through a CICS transaction to basicsoap11provider.xml.

propagatesamlrequester.xml

A service requester pipeline definition that adds configuration information for SAML support with propagation of SAML information through a CICS transaction to basicsoap11requester.xml.

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 provider pipeline configuration file (JSON application handler)

This is a simple example of a configuration file for a service provider pipeline that uses the <cics_json_handler_java> element:

```
<p><?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_json_handler_java>
        <jvmserver>DFHAXIS</jvmserver>
        <repository>/usr/lpp/cicsts/cicsts52/lib/pipeline/repository</repository>
      </cics_json_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

The pipeline contains just one message handler. The handler links to program DFHJSON.

- 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_json_handler_java> element indicates that the pipeline is a Java-based pipeline and the service handler of the pipeline is a message handler that supports JSON messages.
- The <apphandler> element specifies the name of the application handler that the terminal handler of the pipeline links to by default. In this case, the program is DFHJSON, which is the CICS-supplied program for applications deployed with the CICS JSON assistant.

Example provider pipeline configuration file (Channel-attached application handler)

This is a simple example of a configuration file for a service provider pipeline that uses the <cics_soap_1.1_handler> element:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The pipeline contains just one 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> element indicates that the pipeline is not a Java-based pipeline and the terminal handler of the pipeline is a message handler that supports SOAP 1.1 messages.
- The <apphandler> element specifies the name of the application handler that the terminal handler of the pipeline links to by default. In this case, the program is DFHPITP, which is the CICS-supplied program for applications deployed with the CICS web services assistant.

Example provider pipeline configuration file (Axis2 application handler)

This is a simple example of a configuration file for a service provider pipeline that uses the <cics_soap_1.1_handler_java> element:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    <service>
    <terminal_handler>
      <cics_soap_1.1_handler_java>
      <jvmserver>DFHAXIS</jvmserver>
      </cics_soap_1.1_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

The pipeline contains just one 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_java> element indicates that the pipeline is a Java-based pipeline and the service handler of the pipeline is a message handler that supports SOAP 1.1 messages.
- The <apphandler_class> element specifies the supplied Axis2 application handler.

Example requester pipeline configuration file

This is a simple example of a configuration file for a service requester pipeline that uses the <cics_soap_1.2_handler_java> element with Axis2 MTOM/XOP support:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <service>
     <service_handler_list>
       <cics_soap_1.2_handler_java>
         <jvmserver>JVMSERV1</jvmserver>
         <mtom>
       </cics_soap_1.2_handler_java>
     </service_handler_list>
   </service>
</requester_pipeline>
```

The pipeline contains just one message handler.

- The <requester_pipeline> element is the root element of the pipeline configuration file for a service requester pipeline.
- The <service> element specifies the message handlers that are invoked for every request. In the example, there is just one message handler.
- The <service_handler_list> specifies a list of message handlers that are invoked for every request.
- The <cics_soap_1.2_handler_java> element indicates that the pipeline supports Java and the service handler of the pipeline is a message handler that supports SOAP 1.2 messages.
- The <jvmserver> element specifies the JVM server to be used.
- The <mtom/> element specifies that outbound XOP documents are packaged into MTOM messages and sent. By default, inbound MTOM messages are accepted and unpackaged for Java-based pipelines.

Example provider pipeline configuration file for a z/OS Connect for CICS JSON web service

This is a simple example of a configuration file for a service provider pipeline that uses the <provider_pipeline_json> element. Because a java_parser="NO" attribute is supplied, it uses the non-Java JSON parser:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline_json java_parser="NO"
  xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

The <provider_pipeline_json> element differs from the <provider_pipeline> element in that handler programs cannot be defined.

- The <provider_pipeline_json> element is the root element of the pipeline configuration file for a z/OS Connect for CICS JSON web service provider pipeline.
- The java_parser="NO" attribute specifies that the non-Java JSON parser is used.
- The <jvmserver> element specifies the JVM server to be used.

Note: An attempt to start a <provider_pipeline_json> pipeline by using anything other than z/OS Connect for CICS results in an error.

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 WebSphere 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 WebSphere 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 TCPISERVICE(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 z/OS UNIX. 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 25.

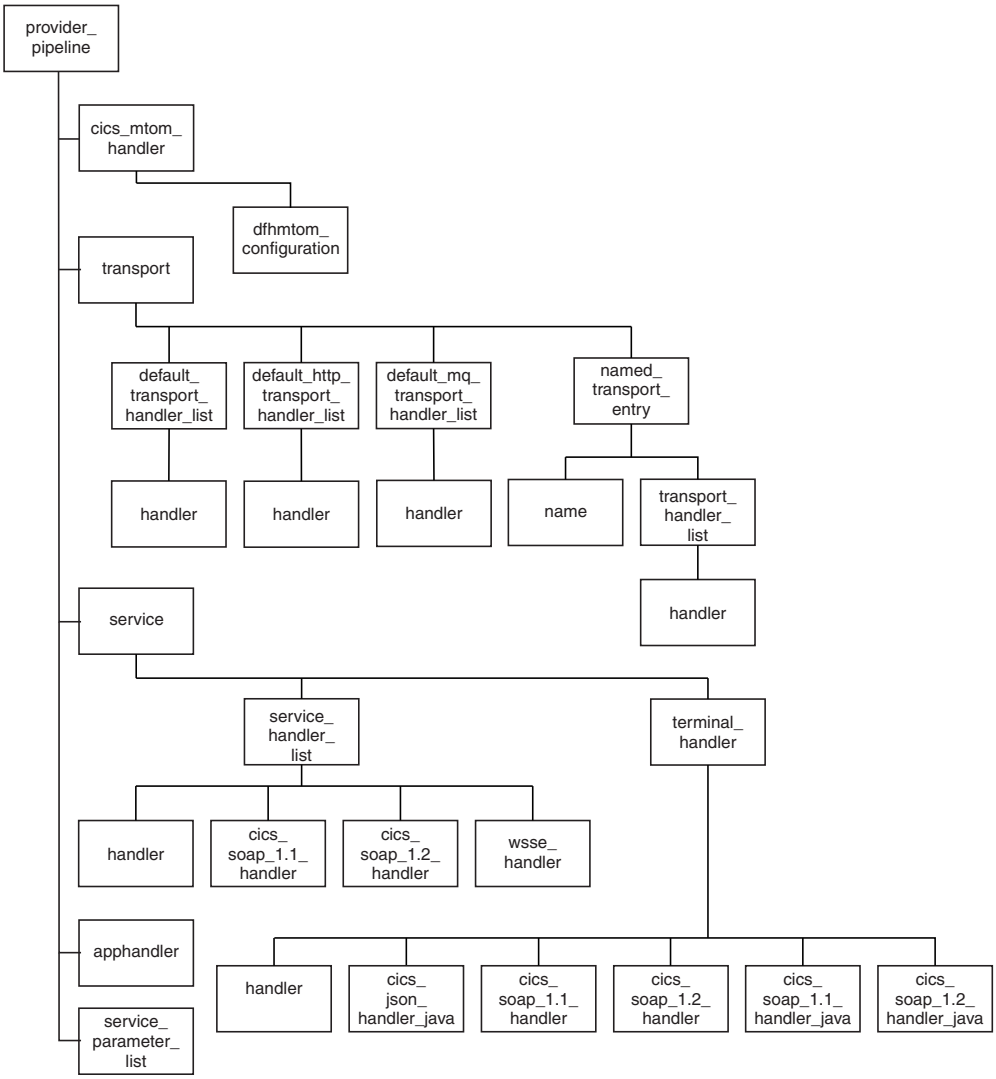


Figure 25. Structure of the pipeline definition for a service provider.

Note: In order to simplify the figure, child elements of the `<handler>`, `<cics_json_handler_java>`, `<cics_soap_1.1_handler>`, `<cics_soap_1.2_handler>`, `<cics_soap_1.1_handler_java>`, and `<cics_soap_1.2_handler_java>` elements are not shown.

The pipeline definition for a service requester

The message handlers are defined in an XML document, which is stored in z/OS UNIX. 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 26.

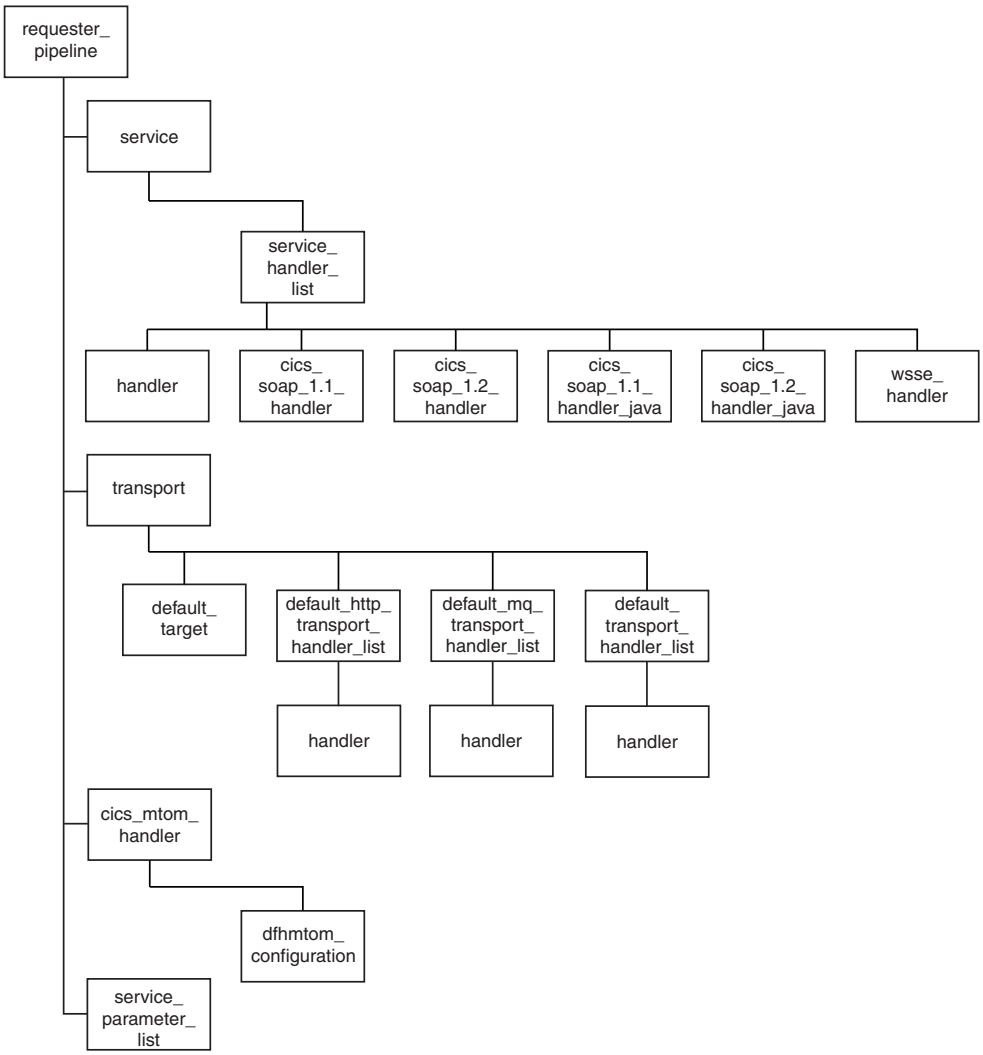


Figure 26. Structure of the pipeline definition for a service requester.

Note: In order to simplify the figure, child elements of the `<handler>`, `<cics_soap_1.1_handler>`, `<cics_soap_1.2_handler>`, `<cics_soap_1.1_handler_java>`, and `<cics_soap_1.2_handler_java>` elements are not shown.

Elements used only in service providers

Some of the XML elements used in a pipeline configuration file apply only to service provider pipelines.

The <apphandler> element

Specifies the name of the application handler that the terminal handler of the pipeline links to by default.

The <apphandler> element is used when the terminal handler is one of the supplied SOAP message handlers. This situation occurs when the <terminal_handler> element contains a <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java>, or <cics_soap_1.2_handler_java> element. However, if your <terminal_handler> element contains a <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> element, you can use the supplied Axis2 application handler by specifying the <apphandler_class> element instead of the <apphandler> element. For more information see the <apphandler_class> element. However, you must not specify <apphandler_class> and <apphandler> elements in the same pipeline configuration file.

If you deploy your web service applications using the CICS web services assistant, you must specify one of the following application handlers in the <apphandler> element.

- The supplied application handler DFHPITP if you do not want to process your application handler using Java.
- Your own application handler that uses DFHPITP.
- The name of the PROGRAM resource that you create.

For more information about application handlers, see "Application handlers" on page 275.

Used in:

- Service provider

Contained by:

- <provider_pipeline> element

Example

```
<apphandler>DFHPITP</apphandler>
```

The <apphandler_class> element

Specifies that the terminal handler of the pipeline links to an Axis2 application handler.

The <apphandler_class> element is used to specify an Axis2 application handler when your <terminal_handler> element contains a <cics_json_handler_java>, <cics_soap_1.1_handler_java>, or <cics_soap_1.2_handler_java> element. To use the supplied Axis2 application handler, specify com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler in the <apphandler_class> element. When using the CICS SOAP handlers, you can also specify your own Axis2 application handler class.

Alternatively, you can specify the <apphandler> element in your pipeline configuration file if you want to use a channel-attached application handler, for more information see the <apphandler> element. However, you must not specify <apphandler_class> and <apphandler> elements in the same pipeline configuration file.

Note: You must not use the <apphandler> element with the <cics_json_handler_java> element.

You must not use the <apphandler_class> element if your <terminal_handler> element contains either a <cics_soap_1.1_handler> or <cics_soap_1.2_handler> element.

For more information about application handlers, see "Application handlers" on page 275.

Used in:

- Service provider

Contained by:

- <provider_pipeline> element

Example

```
<apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
```

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 WebSphere MQ transport, the named resource is the local input queue on which the request is received.
- For the HTTP transport, the resource is the TCPIPSERVICE 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 TCPIPSERVICE

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 TCPIPSERVICE with the name PORT80.

The <provider_pipeline> element

Specifies 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 <cics_mtom_handler> element
2. Optional <transport> element
3. <service> element
4. Optional <apphandler> element
5. Optional <apphandler_class> element
6. 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 <provider_pipeline_json> element

Specifies the root element of the XML document that describes the configuration of the CICS pipeline for a z/OS Connect JSON web service provider.

This differs from the <provider_pipeline> element in that handler programs cannot be defined. This style of pipeline is used as a container for the **WEBSERVICE** resources that are used by z/OS Connect. An attempt to start a <provider_pipeline_json> pipeline using anything other than z/OS Connect will result in an error. The resultant **PIPELINE** resource can't be used as the target of a **USAGE(PIPELINE) URIMAP** resource. It can only be used with **USAGE(JVMSERVER) URIMAP** resources.

Used in:

- Service provider

Attributes:

java_parser={**yes**|**no**}
Select the type of JSON parser that is used to process inbound messages.

Possible values are:

yes Perform the JSON parsing by using Java within the JVM server. This is the default.

no Perform non-Java parsing of the JSON message.

Note: The `java_parser` attribute is optional. If you do not supply it, the default behavior is to parse the JSON message by using Java, within the JVM server. This would be the same as specifying `java_parser="yes"`.

Contains:

- A <jvmserver> element, containing the name of the JVMSERVER resource in which z/OS Connect is configured.

Example that uses Java parsing

```
<provider_pipeline_json java_parser="yes">
  <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

Example that uses non-Java parsing

```
<provider_pipeline_json java_parser="no">
  <jvmserver>DFHWLP</jvmserver>
</provider_pipeline_json>
```

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_json_handler_java>`

`<cics_soap_1.1_handler>`

`<cics_soap_1.2_handler>`

`<cics_soap_1.1_handler_java>`

`<cics_soap_1.2_handler_java>`

If you expect your pipeline to process both SOAP 1.1 and SOAP 1.2 messages, you must use either the `<cics_soap_1.2_handler>` or `<cics_soap_1.2_handler_java>` element.

Remember: In a service provider, you can specify the `<cics_soap_1.1_handler>` and `<cics_soap_1.2_handler>` in the `<service_handler_list>` element, as well as in the `<terminal_handler>` element. However, in a service provider, you can only specify `<cics_soap_1.1_handler_java>` and `<cics_soap_1.2_handler_java>` 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 TCPIPSERVICE 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 <cics_mtom_handler> element
4. 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 service requester pipelines

Some of the XML elements used in a pipeline configuration file apply to both service provider and service requester pipelines.

The <addressing> element

Specifies the support for Web Services Addressing in Java-based SOAP processing.

Used in:

- Service provider
- Service requester

Contained by:

<cics_soap_1.1_handler_java> element

<cics_soap_1.2_handler_java> element

Contains:

A <namespace> element. In a service provider, this element is optional. The element contains one of the two WS-Addressing schemas that are supported by CICS. For inbound messages, Axis2 supports both specifications. For outbound messages, the namespace specified in this element is used. If you do not specify this element or you have two elements, CICS uses the same specification on the outbound message as the inbound message. In a service requester, this element is required and you can specify only one namespace for the outbound message.

This example shows the configuration for a service provider pipeline, where both WS-Addressing specifications are supported. CICS uses the same specification on the outbound message as the inbound message. You can get the same results by specifying an empty <addressing> element.

```
<addressing>
       <namespace>http://www.w3.org/2005/08/addressing</namespace>
       <namespace>http://schemas.xmlsoap.org/ws/2004/08/addressing</namespace>
</addressing>
```

The <cics_json_handler_java> element

Specifies the attributes of the handler program for JSON messages in Java-based JSON pipelines.

Used in:

- Service provider

Contained by:

The <service_handler_list> element

The <terminal_handler> element

Contains:

1. A <jvmserver> element.
2. An optional <repository> element.

Example

The following example shows the XML for the Java-based JSON handler and its nested elements:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_json_handler_java>
        <jvmserver>DFHAXIS</jvmserver>
        <repository>/usr/lpp/cicsts/cicsts53/lib/pipeline/repository</repository>
      </cics_json_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

The <cics_soap_1.1_handler> element

Specifies the attributes of the handler program for SOAP 1.1 messages in non-Java pipelines

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 (Uniform 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 run, but which of the programs will run is not defined.

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_1.1_handler_java> element

Specifies the attributes of the handler program for SOAP 1.1 messages in Java-based SOAP pipelines.

Used in:

- Service requester
- Service provider

Contained by:

<service_handler_list> element

<terminal_handler> element

Contains:

1. A <jvmserver> element.
2. An optional <repository> element.
3. An optional <addressing> element. If you enable Web Services Addressing in Axis2, do not use the DFHWSADH header processing program.
4. Zero, one, or more <headerprogram> elements. Each <headerprogram> element contains:
   a. A <program_name> element, containing the name of a header processing program. You can write Axis2 handlers in Java to process the SOAP headers.
   b. 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 (Uniform Resource Identifier) of the header block's namespace.
   c. 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` and 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>` elements, but that specify different header programs, only one of the header programs will run, but which of the programs will run is not defined.

d. 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

The following example shows the XML for the Java-based SOAP handler and its nested elements:

```
<cics_soap_1.1_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.1_handler_java>
```

The `<cics_soap_1.2_handler>` element

Specifies the attributes of the handler program for SOAP 1.2 messages in non-Java pipelines.

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 (Uniform 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 run, but which of the programs will run is not defined.

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 <cics_soap_1.2_handler_java> element

Specifies the attributes of the handler program for SOAP 1.2 messages in Java-based SOAP pipelines.

Used in:

- Service requester
- Service provider

Contained by:

<service_handler_list> element

<terminal_handler> element

Contains:

1. A <jvmserver> element.
2. An optional <repository> element.

3. An optional <addressing> element. If you enable support for Web Services Addressing in Axis2, do not use header processing programs. You can write Axis2 handlers in Java to process the SOAP headers.
4. Zero, one, or more <headerprogram> elements. Each <headerprogram> element contains:
   a. A <program_name> element, containing the name of a header processing program
   b. 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 (Uniform Resource Identifier) of the header block's namespace.
   c. 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 and 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> elements, but that specify different header programs, only one of the header programs will run, but which of the programs will run is not defined.

d. 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

The following example shows the XML for the Java-based SOAP handler and its nested elements:

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

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 WebSphere 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_parameter_list/>
  </handler>
  <handler>
     <program>HANDLER2</program>
     <handler_parameter_list/>
  </handler>
</default_transport_handler_list>
```

The <handler> element

Specifies 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>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java>, and <cics_soap_1.2_handler_java> 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

```
<?xml version="1.0"?>
<provider_pipeline>
        xmlns="http://www.ibm.com/software/htp/cics/pipeline">
 <service>
  <service_handler_list>
   <handler>
     <program>MYPROG</program>
     <handler_parameter_list><output print="yes"/></handler_parameter_list>
   </handler>
  </service_handler_list>
  <terminal_handler>
   <cics_soap_1.1_handler>
    ...
   </cics_soap_1.1_handler>
  </terminal_handler>
 </service
 <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

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 <jvmserver> element

Specifies the name of the JVMSERVER resource.

This element identifies the name of the JVMSERVER resource, which will process the request. If a value is not supplied, an error message is generated and the PIPELINE is installed in the DISABLED state.

Used in:

- Service provider
- Service requester

Contained by:

- The <cics_json_handler_java> element in Configuring element
- The <cics_soap_1.1_handler_java> element in Configuring element
- The <cics_soap_1.2_handler_java> element in Configuring element
- element

Example

```
<jvmserver>JVMSERVER_NAME</jvmserver>
```

The <repository> element

Specifies the directory name of the Axis2 repository.

This optional element identifies the directory name of the Axis2 repository. If you use this option, you must specify The <jvmserver> element in Configuring beforehand in the handler XML. If the element is not supplied then, the sample repository will be used. When you install CICS Transaction Server the sample

Axis2 repository is installed in the /usr/lpp/cicsts/cicsts53/lib/pipeline/repository directory, where /usr/lpp/cicsts/cicsts53 is the default installation directory for CICS files on z/OS UNIX.

Used in:

- Service provider
- Service requester

Contained by:

- The <cics_json_handler_java> element in Configuring
- The <cics_soap_1.1_handler_java> element in Configuring
- The <cics_soap_1.2_handler_java> element in Configuring

Example

```
<cics_soap_1.1_handler_java>
 <jvmserver>JVMSERV1</jvmserver>
 <repository>/lib/pipeline/repository</repository>
</cics_soap_1.1_handler_java>
```

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:

- <cics_soap_1.1_handler>
- <cics_soap_1.2_handler>
- <cics_soap_1.1_handler_java>
- <cics_soap_1.2_handler_java>
- <handler>
- <wsse_handler>

You determine the order that each handler is called at run time by the order that you specify the handler elements in the <service_handler_list> element. For example, if your pipeline supports WS-Security, encrypted SOAP messages remain encrypted until the <wsse_handler> element is called. Therefore, you must specify the <wsse_handler> element before any other handler program that processes unencrypted messages.

The <service_handler_list> element for a service provider cannot contain the <cics_soap_1.1_handler_java> and <cics_soap_1.2_handler_java> elements, because these elements must be specified in the <terminal_handler> element for Java-based pipelines. A service requestor can contain the <cics_soap_1.1_handler_java> and <cics_soap_1.2_handler_java>, however if these elements are used, they must be the first element listed in the <service_handler_list> element.

If you expect your pipeline to process both SOAP 1.1 and SOAP 1.2 messages, you must use either the <cics_soap_1.2_handler> or <cics_soap_1.2_handler_java> element.

You can use either a SOAP 1.1 or a SOAP 1.2 handler in a service requester pipeline, but in this case the SOAP 1.2 handler does not support SOAP 1.1 messages. Do not specify the SOAP 1.1 or SOAP 1.2 handler in the pipeline if your service requester applications are sending complete SOAP envelopes in the DFHREQUEST container. This avoids duplicating the SOAP message headers in outbound messages.

In a service provider, you can specify the generic handler and SOAP handlers in the <terminal_handler> element as well as in the <service_handler_list> element. For more information about processing SOAP header, see "Header processing programs" on page 285.

Example

```
<service_handler_list>
  <wsse_handler>
    ...
  </wsse_handler>
 <cics_soap_1.1_handler_java>
    ...
 </cics_soap_1.1_handler_java>
 <handler>
    ...
 </handler>
</service_handler_list>
```

The <service_parameter_list> element

Specifies the XML elements that are made available to all the message handlers in the pipeline in container DFH-SERVICEPLIST. This is an optional element.

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 240
The root element of the XML document that describes the configuration of a pipeline in a service requester.

"The <provider_pipeline> element" on page 238
Specifies 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 MTOM/XOP

CICS SOAP pipelines can support the Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications. These specifications define a mechanism for sending and receiving binary data using SOAP, without incurring the overhead of base64 encoding. To enable MTOM support, you must configure your pipelines accordingly.

The <mtom> element

Enables MTOM/XOP support for Java-based pipelines. If this element is defined in the pipeline configuration file, MTOM support is enabled for all inbound and outbound messages. However, if this element is not specified in the pipeline configuration file, then MTOM support is enabled for only inbound messages.

Used in:

- Service provider
- Service requester

Contained by:

<cics_soap_1.1_handler_java>

<cics_soap_1.2_handler_java>

For both provider and requester pipeline configuration files, the <mtom> element should be defined after the optional <addressing> element and before the optional <headerprogram> element.

Example

For a provider or requester mode pipeline, you could specify:

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <addressing></addressing>
  <mtom></mtom>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
```

```
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

The <cics_mtom_handler> element

Enables the supplied MTOM handler program for SOAP pipelines. This program provides support for MTOM MIME multipart/related messages that contain XOP documents and binary attachments. MTOM support is enabled for all inbound messages that are received in the pipeline, but MTOM support for outbound messages is conditionally enabled subject to further options.

Used in:

- Service provider
- Service requester

Contained by:

<provider_pipeline>

<requester_pipeline>

In a provider pipeline configuration file, the <cics_mtom_handler> element should be defined before the <transport> element. At run time, the MTOM handler program needs to unpackage the inbound MTOM message before other handlers including the transport handler process it. It is then invoked as the last handler for the response message, to package an MTOM message to send to the web service requester.

In a requester pipeline configuration file the <cics_mtom_handler> element should be defined after the <transport> element. At run time, the outbound request message is not converted into MTOM format until all other handlers have processed it. It is then invoked as the first handler for the inbound response message to unpackage the MTOM message before other handlers process it and return to the requesting program.

Note: You must not use this handler program with Java-based pipelines. For Java-based pipelines, specify the <mtom> element.

Contains:

<dfhmtom_configuration> element

Default options can be changed using configuration options specified in the <dfhmtom_configuration> element. If you do not want to change the default options, you can use an empty element.

Example

For a provider mode pipeline, you could specify:

```
<provider_pipeline>
    <cics_mtom_handler></cics_mtom_handler>
    <transport>
    ....
    </transport>
    <service>
    ....
    </service>
</provider_pipeline>
```

The <dfhmtom_configuration> element

Specifies configuration information for the supplied MTOM handler program for pipelines that do not support Java. This program provides support for MIME messages that contain XOP documents and binary attachments. If you do not specify any configuration for MTOM, CICS assumes default values.

Used in:

- Service provider
- Service requester

Contained by:

```
<cics_mtom_handler>
```

Attributes:

Name	Description
version	An integer denoting the version of the configuration information. The only valid value is 1.

Contains:

- An optional <mtom_options> element
- An optional <xop_options> element
- An optional <mime_options> element

Example

```
<dfhmtom_configuration version="1">
  <mtom_options send_mtom="same" send_when_no_xop="no"/>
  <xop_options apphandler_supports_xop="yes"/>
  <mime_options content_id_domain="example.org"/>
</dfhmtom_configuration>
```

The <mtom_options> element

Specifies when to use MTOM for outbound SOAP messages for pipelines that do not support Java.

Used in:

- Service provider
- Service requester

Contained by:

```
<dfhmtom_configuration>
```

Attributes:

Attribute	Description
send_mtom	Specifies if MTOM should be used to convert the outbound SOAP message into a MIME message: **no** MTOM is not used for outbound SOAP messages. **same** In service provider mode, MTOM is used for SOAP response messages whenever the requester uses MTOM. This is the default value in a service provider pipeline. In service requester mode, specifying this value is the same as when you specify `send_mtom="yes"`. **yes** MTOM is used for all outbound SOAP messages. This is the default value in a service requester pipeline.
send_when_no_xop	Specifies if an MTOM message should be sent, even when there are no binary attachments present in the message. **no** MTOM is only used when binary attachments are being sent with the message. **yes** MTOM is used for all outbound SOAP messages, even when there are no binary attachments to send in the message. This is the default value, and is primarily used as an indicator to the receiving program that the sender supports MTOM/XOP. This attribute can be combined with any of the **send_mtom** attribute values, but has no effect if you specify `send_mtom="no"`.

Example

```
<provider_pipeline>
 <cics_mtom_handler>
  <dfhmtom_configuration version="1">
    <mtom_options send_mtom="same" send_when_no_xop="no"/>
  </dfhmtom_configuration>
 </cics_mtom_handler>
...
</provider_pipeline>
```

In this provider pipeline example, SOAP messages are converted into MTOM messages only when binary attachments need to be sent with the message, and the service requester sent an MTOM message.

The <xop_options> element

Specifies whether XOP processing can take place in direct or compatibility mode for pipelines that do not support Java.

Used in:

- Service provider
- Service requester

Contained by:

`<dfhmtom_configuration>`

Attributes:

Attribute	Description
apphandler_supports_xop	In provider mode, specifies if the application handler is capable of handling XOP documents in direct mode: **no** The application handler cannot handle XOP documents directly. This is the default value if the <apphandler> element does not specify DFHPITP. Compatibility mode is used in the pipeline to handle any inbound or outbound messages that are received or sent in MTOM format. **yes** The application handler can handle XOP documents. This is the default value if the <apphandler> element specifies DFHPITP. Direct mode is used in the pipeline to handle any inbound or outbound messages that are received or sent in MTOM format. This is subject to restrictions at run time. For example, if you have specified WS-Security related elements in the pipeline configuration file, the MTOM handler determines that the pipeline should use compatibility mode rather than direct mode for processing XOP documents. In requester mode, specifies if service requester applications use the CICS web services support to create and handle XOP documents in direct mode. **no** Service requester applications do not use the CICS web services support. Specify this value if your requester application links to DFHPIRT to drive the pipeline, and is therefore not capable of creating and handling XOP documents in direct mode. **yes** Service requester applications do use the CICS web services support. Specify this value if your requester application uses the **EXEC CICS INVOKE WEBSERVICE** command.

Example

```
<provider_pipeline>
 <cics_mtom_handler>
  <dfhmtom_configuration version="1">
    <xop_options apphandler_supports_xop="no"/>
  </dfhmtom_configuration>
 </cics_mtom_handler>
 ...
</provider_pipeline>
```

In this provider pipeline example, inbound MTOM messages and outbound response messages are processed in the pipeline using compatibility mode.

The <mime_options> element

Specifies the domain name that should be used when generating MIME content-ID values for pipelines that do not support Java. The MIME content-ID values are used to identify binary attachments.

Used in:

- Service provider
- Service requester

Contained by:

```
<dfhmtom_configuration>
```

Attributes:

Attribute	Description
content_id_domain	The syntax to use is *domain.name*. To conform to Internet standards, the name should be a valid internet host name and should be unique to the CICS system where the pipeline is installed. Note that this is not checked by CICS. If this element is omitted, CICS uses the value `cicsts`.

Example

```
<provider_pipeline>
<dfhmtom_configuration version="1">
  <mime_options content_id_domain="example.org"/>
</dfhmtom_configuration>
...
</provider_pipeline>
```

In this example, references to binary attachments are created using `cid:`*`unique_value`*`@example.org`.

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 DFHWSSE, and by providing configuration information for the handler.

Example

A provider pipeline configuration file that uses WS-Security might take the following form:

```
<?xml version="1.0"?>
<provider_pipeline
         xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <service>
      <service_handler_list>
         <wsse_handler>
            <dfhwsse_configuration version="1">
               <authentication trust="blind" mode="basic"/>
            </dfhwsse_configuration>
         </wsse_handler>
         <handler>
          ...
         </handler>
      </service_handler_list>
      <terminal_handler>
         <cics_soap_1.2_handler/>
```

```
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

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.

In a provider pipeline configuration file, the CICS supplied message handler for WS-Security might have to decrypt an encypted message. The <wsse_handler> element must be defined before any other handler programs that need to process the unencrypted message content.

In a requester pipeline configuration file, the CICS supplied message handler for WS-Security might have to encrypt a message. It must be defined after any other handler programs that need to process the unencrypted message content, including the CICS SOAP handler program.

The <dfhwsse_configuration> element

Specifies configuration information for the security handler DFHWSSE1, which provides support for securing web services.

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. Either of the following elements:
   - An optional <authentication> element.
     - In a service requester pipeline, the <authentication> element specifies the type of authentication that must be used in the security header of outbound SOAP messages.

- In a service provider pipeline, the element specifies whether CICS uses the security tokens in an inbound SOAP message to determine the user ID under which work is processed.

- An optional <sts_authentication> element.

  The action attribute on this element specifies what type of request to send to the Security Token Service. If the request is to issue an identity token, then CICS uses the values in the nested elements to request an identity token of the specified type.

2. If you specify an <sts_authentication> element, you must also specify an <sts_endpoint> element.

   When this element is present, CICS uses the URI in the <endpoint> element to send a request to the Security Token Service.

3. 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.

4. 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.

5. 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.

6. 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.

7. In provider pipelines only, an optional <reject_signature/> element.

   If this element is present, CICS rejects any message that includes a certificate in its header that signs part or all of the message body. A SOAP fault is issued to the web service requester.

8. In provider pipelines only, an optional <reject_encryption/> element.

   If this element is present, CICS rejects any message that is partially or fully encrypted. A SOAP fault is issued to the web service requester.

Example

```
<dfhwsse_configuration version="1">
  <sts_authentication action="issue">
    <auth_token_type>
      <namespace>http://example.org.tokens</namespace>
      <element>UsernameToken</element>
    </auth_token_type>
    <suppress/>
  </sts_authentication>
  <sts_endpoint>
    <endpoint>https://example.com/SecurityTokenService</endpoint>
  </sts_endpoint>
  <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 and mode	Taken together, the trust and mode attributes specify: • whether asserted identity is used • the combination of security tokens that are used in SOAP messages. Asserted identity allows a trusted user to assert that work must run under a different identity, the *asserted identity*, without the trusted user having the credentials that are associated with that identity. When asserted identity is used, messages contain a *trust token* and an *identity token*. The trust token is used to check that the sender has the correct permissions to assert identities. The identity token holds the asserted identity, that is, the user ID under which the request is run. 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. The allowable combinations of these attributes, and their meanings, are described in Table 4 and Table 5 on page 266.

Table 4. The **mode** *and* **trust** *attributes in a service requester pipeline*

trust	mode	Meaning
none	none	No credentials are added to the message
none	basic	*Invalid combination of attribute values*
none	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.
blind	none	*Invalid combination of attribute values*

Table 4. The **mode** *and* **trust** *attributes in a service requester pipeline (continued)*

trust	mode	Meaning
blind	basic	Asserted identity is not used. CICS adds an identity token to the message, but does not provide a trust token. The identity token is a user name with no password. 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).
blind	signature	*Invalid combination of attribute values*
basic	none	*Invalid combination of attribute values*
basic	basic	*Invalid combination of attribute values*
basic	signature	*Invalid combination of attribute values*
signature	none	*Invalid combination of attribute values*
signature	basic	Asserted identity is used. CICS adds the following tokens to the message: • The trust token is an X.509 security token. • The identity token is a user name with no password. The certificate that is 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).
signature	signature	*Invalid combination of attribute values*

Table 5. 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 checks that any signed elements are correctly signed.
none	basic	Inbound messages must contain a user name security token with a password. CICS puts the user name in the DFHWS-USERID container.
none	basic-ICRX	*Invalid combination of attribute values*
none	basic-kerberos	*Invalid combination of attribute values*
none	signature	Inbound messages must contain an X.509 security token that has been used to sign the message body.
blind	none	*Invalid combination of attribute values*
blind	basic	Inbound messages must contain an identity token, where the identity token contains a user ID and optionally a password. CICS puts the user ID in the DFHWS-USERID container. If no password is included, CICS uses the user ID without verifying it. If a password is included, the security handler DFHWSSE1 verifies it.

Table 5. The **mode** *and* **trust** *attributes in a service provider pipeline (continued)*

trust	mode	Meaning
blind	basic-ICRX	Inbound messages must contain an ICRX identity token. CICS resolves the identity, puts the user ID in the DFHWS-USERID container, and puts the ICRX in container DFHWS-ICRX. Authentication, if required, uses client-certified SSL or another security protocol.
blind	basic-kerberos	*Invalid combination of attribute values*
blind	signature	Inbound messages must contain an identity token, where the identity token is the first X.509 certificate in the SOAP message header. The certificate does not need to have signed the message. The security handler extracts the matching user ID and places it in the DFHWS-USERID container.
basic	none	*Invalid combination of attribute values*
basic	basic	Inbound messages must use asserted identity: • The trust token is a user name token with a password • The identity token is a second user name token without a password. CICS puts this user name in container DFHWS-USERID.
basic	basic-ICRX	Inbound messages must use asserted identity: • The trust token is a user name token with a password. CICS establishes whether the user ID and password combination are valid, and, if they are valid, CICS resolves the asserted ICRX-based identity to a user ID. CICS then performs a surrogate security check from the authenticated identity to the asserted identity. • The identity token is an ICRX, which identifies the specific client user. CICS puts the user name in container DFHWS-USERID and the ICRX in container DFHWS-ICRX.
basic	basic-kerberos	Inbound messages must use asserted identity. One token is required, a Kerberos Version 5 token with one of the following format types: • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510 • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510 • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120 • http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120 The token must be Base-64 encoded. CICS validates the token by using Network Authentication Service for z/OS and puts the user ID associated with the token in container DFHWS-USERID.

Table 5. The **mode** *and* **trust** *attributes in a service provider pipeline (continued)*

trust	mode	Meaning
basic	signature	Inbound messages must use asserted identity: • The trust token is a user name 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.
signature	none	*Invalid combination of attribute values*
signature	basic	Inbound messages must use asserted identity: • The trust token is an X.509 certificate • The identity token is a user name token without a password. CICS puts the user name in container DFHWS-USERID. The identity token and the body must be signed with the X.509 certificate.
signature	basic-ICRX	Inbound messages must use asserted identity. • The trust token is an ICRX signed with an X.509 certificate. CICS resolves the X.509 certificate to a user ID and ensures that the XML signature is valid. CICS resolves the asserted ICRX-based identity to a user ID. CICS then performs a surrogate security check from the authenticated X.509 identity to the asserted ICRX identity. • The identity token is a user name token without a password. CICS puts the user name in container DFHWS-USERID and the ICRX in container DFHWS-ICRX.
signature	basic-kerberos	*Invalid combination of attribute values*
signature	signature	Inbound messages must use asserted identity: • 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. The identity token and the body must be signed with the first X.509 certificate (the trust token).

Notes:

1. The combinations of the trust and mode attribute values are checked when the PIPELINE is installed. The installation fails if the attributes are incorrectly coded.
2. CICS uses password verification to verify a user ID during the processes described here. By default, password verification does not cause RACF® to record that the user ID has been used for a sign-on. If you require this information for audit purposes, or if your system is set up to revoke unused user IDs, specify the system initialization parameter **SECVFYFREQ=USRDELAY** for the CICS region. When you set this system initialization parameter, CICS enforces a full verification request at least once a day for each user ID that is used to log on to the CICS region. The full verification request makes RACF record the date and time of last access for the user ID, and write user statistics.

Contains:

1. An optional, empty <suppress/> element.

   If this element is specified in a service provider pipeline, the handler does not attempt to use any security tokens in the message to determine under which user ID the work runs.

   If this element is specified in a service requester pipeline, the handler does not attempt to add to the outbound SOAP message any of the security tokens that are required for authentication.
2. In a requester pipeline, an optional <algorithm> element that specifies the URI of the algorithm that is used to sign the body of the SOAP message. You must specify this element if the combination of trust and mode attribute values indicate that the messages are signed. You can specify only the RSA with SHA1 algorithm in this element. The URI is http://www.w3.org/2000/09/xmldsig#rsa-sha1.
3. An optional <certificate_label> element that specifies the label that is associated with an X.509 digital certificate installed in RACF. If you specify this element 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.

   This element is ignored in a service provider pipeline.

Example

```
<authentication trust="signature" mode="basic">
  <suppress/>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>AUTHCERT03</certificate_label>
</authentication>
```

The <sts_authentication> element

Specifies that a Security Token Service (STS) must be used for authentication and determines what type of request is sent.

Used in:

- Service provider
- Service requester

Contained by:

<dfhwsse_configuration>

Attributes:

Name	Description
action	Specifies what type of request CICS sends to the STS when a message is received in the service provider pipeline. Valid values are as follows: **issue** The STS issues an identity token for the SOAP message. This value is not valid for SAML in a provider pipeline. **validate** The STS validates the provided identity token and returns whether the token is valid to the security handler. If you do not specify this attribute, CICS assumes that the action is to request an identity token. In a service requester pipeline, you cannot specify this attribute because CICS always requests that the STS issues a token.
extract	This attribute is valid only when you are using SAML. Are the elements of the SAML token to be extracted? Valid values are as follows: **no** The elements of the SAML token are not extracted into containers. **yes** The main elements of the SAML token are extracted and placed in containers that are created by CICS.
token_signature	This attribute is valid only when you are using SAML. Must a token signature be supplied? Valid values are as follows: **ignored** Any signature that is supplied is ignored. **required** A valid signature must be supplied. This is the default value.

Name	Description
tran_channel	This attribute is valid only when you are using SAML. In a service provider pipeline, this attribute specifies whether SAML assertions contained in a message that is received in the pipeline are made available to the target application program in containers in the transaction channel DFHTRANSACTION. Valid values are as follows: **yes** The SAML assertions are copied into containers in the DFHTRANSACTION channel to be made available to the program. For more information about container names and types, see The SAML linkable interface, DFHSAML. **no** SAML assertions are not made available to the program via the DFHTRANSACTION channel, but in containers in the channel that is passed to the program by the pipeline. This is the default value. If you do not specify this attribute for a service provider, the assertions are made available only in containers in the channel that is passed to the program from the SOAP pipeline. In a service requester pipeline, this attribute specifies whether the SAML token contained in the DFHSAML-OUTTOKEN container of the transaction channel DFHTRANSACTION is used on the request. Valid values are as follows: **yes** The contents of the DFHTRANSACTION channel's DFHSAML-OUTTOKEN container are used as the SAML token for the request. **no** The contents of the DFHSAML-OUTTOKEN container in the channel that is passed to the pipeline are used as the request's SAML token. This is the default value. If you do not specify this attribute for a service requester, the SAML token is taken from the DFHSAML-OUTTOKEN container in the channel that is passed to the SOAP pipeline.

Contains:

1. An `<auth_token_type>` element. This element is required when you specify a `<sts_authentication>` element in a service requester pipeline and is optional in a service provider pipeline. For more information, see `<auth_token_type>`.
   - In a service requester pipeline, the `<auth_token_type>` element indicates the type of token that STS issues when CICS sends it the user ID contained in the DFHWS-USERID container. The token that CICS receives from the STS is placed in the header of the outbound message.
   - In a service provider pipeline, the `<auth_token_type>` element is used to determine the identity token that CICS takes from the message header and sends to the STS to exchange or validate. CICS uses the first identity token of the specified type in the message header. If you do not specify this element, CICS uses the first identity token that it finds in the message header. CICS does not consider the following as identity tokens:
     - `wsu:Timestamp`

- xenc:ReferenceList
- xenc:EncryptedKey
- ds:Signature

2. In a service provider pipeline only, an optional, empty <suppress/> element. If this element is specified, the handler does not attempt to use any security tokens in the message to determine the user ID that the work runs under. The <suppress/> element includes the identity token that is returned by the STS.

Example

The following example shows a service provider pipeline, where the security handler requests a token from the STS.

```
<sts_authentication action="issue">
   <auth_token_type>
      <namespace>http://example.org.tokens</namespace>
      <element>UsernameToken</element>
   </auth_token_type>
   <suppress/>
</sts_authentication>
```

The <auth_token_type> element

Specifies what type of identity token is required.

This element is mandatory when you specify the <sts_authentication> element in a service requester pipeline, and optional in a service provider.

- In a service requester pipeline, the <auth_token_type> element indicates the type of token that STS issues when CICS sends it the user ID contained in the DFHWS-USERID container. The token that CICS receives from the STS is placed in the header of the outbound message.
- In a service provider pipeline, the <auth_token_type> element is used to determine the identity token that CICS takes from the message header and sends to the STS to exchange or validate. CICS uses the first identity token of the specified type in the message header. If you do not specify this element, CICS uses the first identity token that it finds in the message header. CICS does not consider the following as identity tokens:
  - wsu:Timestamp
  - xenc:ReferenceList
  - xenc:EncryptedKey
  - ds:Signature

Used in:

- Service provider
- Service requester

Contained by:

<sts_authentication>

Contains:

1. A <namespace> element. This element contains the namespace of the token type that is to be validated or exchanged.

   If you are using SAML, set the content of this element to either urn:oasis:names:tc:SAML:1.0:assertion or urn:oasis:names:tc:SAML:2.0:assertion, depending on the version of SAML.

2. An <element> element. This element contains the local name of the token type that is to be validated or exchanged.

   For SAML, use the local name Assertion.

The values of these elements form the Qname of the token.

Example

```
<auth_token_type>
   <namespace>http://example.org.tokens</namespace>
   <element>UsernameToken</element>
</auth_token_type>
```

The <sts_endpoint> element

Specifies the location of the Security Token Service (STS).

Used in:

- Service provider
- Service requester

Contained by:

<dfhwsse_configuration>

Contains:

- An <endpoint> element. This element contains a URI that points to the location of the Security Token Service (STS) on the network. It is recommended that you use SSL or TLS to keep the connection to the STS secure, rather than using HTTP.

  To use SAML support, set the endpoint to cics://PROGRAM/DFHSAML.

  You can also specify a WebSphere MQ endpoint, by using the JMS format of URI.
- An optional <jvmserver> element. This element identifies the JVM server that is configured to run the SAML token service. If this element is not included, the default sample resource JVM server DFHXSTS is assumed. This element is valid only if you are using SAML: if you use it in other situations, an error occurs.

Examples

In this example, the endpoint is configured to use a secure connection to the STS at the specified URI.

```
<sts_endpoint>
   <endpoint>https://example.com/SecurityTokenService</endpoint>
</sts_endpoint>
```

In this example, the endpoint is configured to use CICS SAML support.

```
<sts_endpoint>
   <endpoint>cics://PROGRAM/DFHSAML</endpoint>
</sts_endpoint>
```

The <sign_body> element

Directs DFHWSSE 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
Digital Signature Algorithm with Secure Hash Algorithm 1 (DSA with SHA1) Supported on inbound SOAP messages only.	`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`

2. A `<certificate_label>` element that specifies the label associated with a digital certificate installed in RACF. The digital certificate provides the key that is used to sign the message.

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 DFHWSSE 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`

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

2. A <certificate_label> element that specifies the label that is associated with a digital certificate in RACF. The digital certificate provides the key that is used to encrypt the message.

Example

```
<encrypt_body>
  <algorithm>http://www.w3.org/2001/04/xmlenc#aes256-cbc</algorithm>
  <certificate_label>ENCCERT02</certificate_label>
</encrypt_body>
```

Application handlers

An application handler is a CICS program that the terminal handler of a SOAP service provider pipeline links to at run time.

Application handlers are used in provider mode pipelines in which the terminal handler is one of the supplied SOAP message handlers. This situation occurs when the <terminal_handler> element contains a <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java> or a <cics_soap_1.2_handler_java> element.

The application handler is responsible for processing the body of a SOAP request, and for generating a response using the returned data. The application handler can call other programs to complete this processing. Typically the application handler acts as a general-purpose presentation layer around one or more business applications. It is responsible for mapping XML into a form that an application can use, attaching that application, and then generating a response using the data returned.

An application handler can be attached by CICS in two ways. The typical mechanism involves a channel and control containers; the other method involves Java bindings for Axis2.

Channel-attached application handlers are specified in the <apphandler> element of the <provider_pipeline> element. At run time, the DFHWS-APPHANDLER container is populated by the contents of <apphandler>. However, the DFHWS-APPHANDLER container can be dynamically updated by any of the other message handlers. Therefore, the program that is linked to at run time can be different to the program specified in the <apphandler> element. The following application handlers can be specified in the <apphandler> element or the DFHWS-APPHANDLER container:

- The supplied channel-attached SOAP application handler, DFHPITP. For more information about channel-attached application handlers, see "Channel-attached application handlers" on page 276

- Your own channel-attached application handler. This application handler can be written in languages other than Java. For more information about the control containers that can be used in your channel-attached application handler, see "Control containers" on page 291.
- Your own Java application handler for Java-based pipelines, which implements the ApplicationHandler Java interface and that is attached to the pipeline using Axis2 MessageContext. For more information about the ApplicationHandler Java interface, see JCICS class reference in Reference -> Application development reference.

To use an application handler that uses Java bindings for Axis2, you must specify the `<apphandler_class>` element of the `<provider_pipeline>` element. Axis2 application handlers also require that a JVM server must exist for the web services pipeline and application handler to run on and that the terminal handler of your web services pipeline must be either the `<cics_soap_1.1_handler_java>` or the `<cics_soap_1.2_handler_java>` message handler. To use the supplied Axis2 application handler, you must specify `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler` in the `<apphandler_class>` element, however you can specify your own Axis2 application handler class. At run time, the `DFHWS-APPHANCLAS` container is populated by the contents of `<apphandler_class>`.

For web service applications that are deployed using the CICS web services assistant, you must specify either DFHPITP or your own application handler that uses DFHPITP in the `<apphandler>` element, or specify `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler` in the `<apphandler_class>` element. For more information about the CICS web services assistant, see "The CICS web services assistant" on page 52.

It is also possible to deploy Axis2 applications as provider mode web services in CICS using the Axis2 style of web service deployment. For more information, see "Deploying a Java provider-mode web service in an Axis2 JVM server" on page 146.

Channel-attached application handlers

Channel-attached application handlers are application handlers that are attached to CICS using a channel and control containers.

The channel that is used by the application handler is the DFHAHC-V1 channel. This channel passes the following containers between the terminal handler and the provider-mode web service application:

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. If the application does not modify the contents, the application handler program must update the contents of this container, even if it is putting the same content back into the container before returning to the terminal handler.

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 terminal handler. For example, a header processing program can add containers to the channel. These containers are passed as user containers. For more information about application handlers, see "Application handlers" on page 275.

Message handlers

A message handler is a CICS program that 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 or JSON request or response, without changing it
- Change the contents of an XML or JSON request or response
- In a non-terminal message handler, pass an XML or JSON 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 SOAP handlers, <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java>, 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 that are used as message handlers are invoked with the same interface: they are invoked with a *channel* that holds a number of containers. The containers can be categorized as the following types:

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* that are passed to the programs.

Some of the context containers hold information that 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 that 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 that start 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:

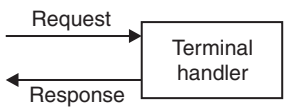
- Some message handlers do not change the message contents, nor do they change the normal processing sequence of a pipeline
- Some message handlers change the message contents, but do not change the normal processing sequence of a pipeline
- Some message handlers change the processing sequence of a pipeline.

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.

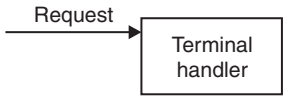
Terminal handler protocols

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.

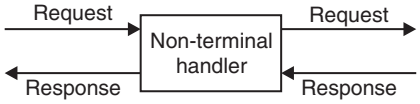
Normal request, with no response
This is another common protocol for a terminal handler.



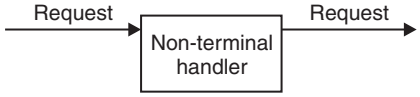
This protocol is usually encountered when the target application determines that there should be no response to the request (although the decision may also be made in the terminal handler).

Non-terminal handler protocols

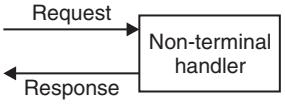
Normal request and response
This is the usual protocol for a non-terminal handler. The handler is invoked for a request message, and again for the response message. In each case, the handler processes the message, and passes it to the next handler in the pipeline.



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.

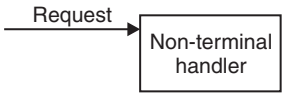


Handler creates the response
This protocol is typically used in abnormal situations, because the non-terminal handler does not pass the request to the next handler. Instead it constructs a response, and returns it to the pipeline.



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.

About this task

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.

Procedure

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.

About this task

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

Procedure

1. Using the contents of container DFHFUNCTION, determine if the message passed to this message handler is a request or a response.

DFHFUNCTION	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 DFHFUNCTION contains PROCESS-REQUEST, the message handler is a terminal message handler, and these steps do not apply.
   - If DFHFUNCTION 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 makes no changes to any of the containers passed to it.

   It is an error for a message handler program to return any of the following:
   - An empty DFHRESPONSE container.
   - A non-empty DFHREQUEST container and a non-empty DFHRESPONSE container.
   - An empty DFHREQUEST container on the outbound request.

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.

Procedure

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.

Results

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.

Procedure

1. Delete container DFHREQUEST.
2. Construct your response, and put it in container DFHRESPONSE.

Results

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.

Procedure

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.

About this task

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.

About this task

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.

Procedure

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.

Results

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 might occur in the pipeline.

About this task

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 must write your handler program to handle any errors that might occur.

Procedure

1. Check that container DFHFUNCTION contains HANDLER-ERROR, indicating that the message handler has been called for error processing.

   Tip:
   - If DFHFUNCTION 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 "DFHERROR container" on page 291.

   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).

Results

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.

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 that 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. For pipelines that do not support Java, the `<cics_soap_1.1_handler>` or `<cics_soap_1.2_handler>` message handlers must be specified. For pipelines that support Java, the `<cics_soap_1.1_handler_java>`, or `<cics_soap_1.2_handler_java>` message handlers must be specified.

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.

You must not define `<cics_soap_1.1_handler>` and `<cics_soap_1.2_handler>` message handlers, or `<cics_soap_1.1_handler_java>` and `<cics_soap_1.2_handler_java>` message handlers in the same pipeline. If you expect your pipeline to process both SOAP 1.1 and SOAP 1.2 messages, you should use either the `<cics_soap_1.2_handler>` or `<cics_soap_1.2_handler_java>` message handler.

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 that 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 that the header program can examine or modify, including the SOAP header block for which the program is invoked, and the SOAP message body.

The channel and the containers that the header processing program can use are described in “The header processing program interface” on page 287.

Other containers hold information about the environment in which the header program is invoked, for example:

- 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>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java>, and <cics_soap_1.2_handler_java> 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 following items:

- 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 are selected for processing by the remaining rules:

- If none of the header blocks are selected, the header processing program is invoked once.
- If any of the header blocks are 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 following conditions are satisfied:

- The name of the root element of the header block matches the <localname> element in the pipeline configuration file
- The namespace of the root element 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 is selected for processing when the following lines are coded in the pipeline configuration file:

```
<namespace>http://mynamespace</namespace>
<localname>myheaderblock</localname>
```

The `<localname>` elements can contain an `*` to indicate that all header blocks in the namespace should be processed. Therefore, the same header block is selected by the following code:

```
<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 are 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 is 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, and DFHXMLSS-PARSE.

Note: Although web service that use Axis2 to process SOAP messages can use the header processing program interface, it is more efficient to write your own Axis2 handlers in Java to process the SOAP headers. For more information on creating Axis2 handlers, see Writing Your Own Axis2 Module

Container DFHHEADER

When the header processing program is called, DFHHEADER contains the single header block that 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 called even when there is no matching header block in the SOAP message. In this case, container DFHHEADER has a length of zero. This is the case when a header processing program is called 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 called, by specifying `<mandatory>true</mandatory>` or `<mandatory>1</mandatory>`.

When the header program returns, container DFHHEADER must contain zero, one, or more header blocks that 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 called, DFHWS-XMLNS contains information about XML namespaces that are declared in the SOAP envelope. The header program can use this information to perform the following tasks:

- Resolve qualified names that it encounters in the header block
- 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 that are not visible in the program may rely on them.

Container DFHWS-BODY

This container contains the body section of the SOAP envelope. The header processing program can modify the contents.

When the header processing program is called, 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.

Container DFHXMLSS-PARSE

When you use either the <cics_soap_1.1_handler> or <cics_soap_1.2_handler> elements in your pipeline configuration, and header program is called, DFHXMLSS-PARSE contains the XML System Services (XMLSS) records for that header. This container is not created when <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> elements are used.

Control, context, and user containers

As well as the containers described, the interface passes the *control containers*, *context containers*, and *user containers* on channel DFHHHC-V1.

For more information about these containers, see "Containers used in the pipeline" on page 290.

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 handler program specified in container DFHWS-APPHANDLER is, in some cases, eligible for dynamic routing. All pipeline processing before the application handler program is always performed locally in the CICS region that received the SOAP message.

The transaction that runs the target application handler program is eligible for routing when one of the following conditions is true:

- The transaction under which the pipeline is processing the message is defined as DYNAMIC or REMOTE. This transaction is defined in the URIMAP that is used to map the URI from the inbound SOAP message.
- 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.
- A WS-AT SOAP header exists in the inbound SOAP message.

In all the preceding scenarios, a task switch occurs during the pipeline processing. The second task runs under the transaction specified in the DFHWS-TRANID container. This task switch provides an opportunity for dynamic routing to take place, but only if MRO is used to connect the CICS regions together. In addition, the CICS region that you are routing to must support channels and containers.

The routing only takes 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 about the routing of web service requests, see technote: *Routing of provider mode CICS web services*.

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 users program. The request is then 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 V3.1 or higher. If you are using a COMMAREA, this restriction does not apply.

"Daisy-chaining" is not supported. That is, once a request has been dynamically routed to a target region it cannot be dynamically routed from the target to a third region, even though the transaction is defined as ROUTABLE(YES) and DYNAMIC(YES). The transaction can, however, be statically routed from the target region to a third region.

For more information, see the *CICS Customization Guide*.

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 that has 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 that 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 in these ways:

Control containers
These 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. The names of the control containers are defined by CICS, and begin with the characters DFH.

Context containers
These containers 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 to 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 characters DFH.

Header processing program containers
These containers contain information that is used by header processing programs that are called from the CICS-supplied SOAP message handlers. For information about these containers, see The header processing program interface.

Security containers
These containers contain information that is used by the Trust client interface and the security message handler to process security tokens using a Security Token Service (STS). The names of the security containers are defined by CICS, and begin with the characters DFH.

Generated containers
These containers contain the data from the SOAP message, such as variable arrays and long strings, that is passed to and from the application program for processing. CICS automatically creates these containers during pipeline processing, and the names begin with the characters DFH.

User containers
These containers contain information that 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.

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.

DFHERROR container

DFHERROR is a container of DATATYPE(BIT) that is used to convey information about pipeline errors to other message handlers.

Table 6. Structure of the DFHERROR container.

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 7 on page 292.
PIISNEB-ERROR-MODE	1	**P** The error occurred in a provider pipeline **R** The error occurred in a requester pipeline **T** The error occurred in a Trust client
PIISNEB-ABCODE	4	The abend code when the error is associated with a transaction abend.

Table 6. Structure of the DFHERROR container. (continued)

Field name	Length (bytes)	Contents
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 7. Values for the PIISNEB-ERROR-TYPE field

Value of PIISNEB-ERROR-TYPE	Meaning
1	The handler program failed. 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 the target program failed, the abend code is in container PIISNEB-ABCODE.
6	The pipeline manager failed to communicate with a remote server because of an error in the underlying transport.
7	The DFHWS-STSACTION container has an error. It is missing, corrupt, or contains an incorrect 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 following table shows the language copybooks that map the container.

Table 8. Copybooks that map the container

Language	Copybook
COBOL	DFHPIUCO
PL/I	DFHPIUCL
C and C++	dfhpiuch.h
Assembler	DFHPIUCD

DFHFUNCTION container

DFHFUNCTION is a container of DATATYPE(CHAR) that contains a 16-character string that indicates where in a pipeline a program is being called.

The string has one of the following values. The rightmost character positions are padded with blank characters.

RECEIVE-REQUEST
The handler is a nonterminal handler in a service provider pipeline, and is being called to process an inbound request message. On entry to the handler, the message is in control container DFHREQUEST.

SEND-RESPONSE
The handler is a nonterminal handler in a service provider pipeline, and is being called to process an outbound response message. On entry to the handler, the message is in control container DFHRESPONSE.

SEND-REQUEST
The handler is being called 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 called 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 called as the terminal handler of a service provider pipeline

NO-RESPONSE
The handler is being called after processing a request, when no response is to be processed.

HANDLER-ERROR
The handler is being called because an error has been detected.

In a service provider pipeline that processes a request and returns a response, the values of DFHFUNCTION that occur are `RECEIVE-REQUEST`, `PROCESS-REQUEST`, and `SEND-RESPONSE`. Figure 27 on page 294 shows the sequence in which the handlers are called and the values of DFHFUNCTION that are passed to each handler.

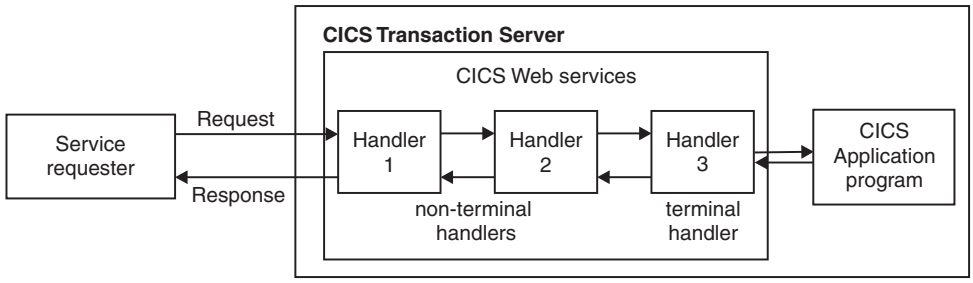


Figure 27. Sequence of handlers in a service provider pipeline

Sequence	Handler	DFHFUNCTION
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 DFHFUNCTION that occur are SEND-REQUEST and RECEIVE-RESPONSE. Figure 28 shows the sequence in which the handlers are called, and the values of DFHFUNCTION that are passed to each handler.

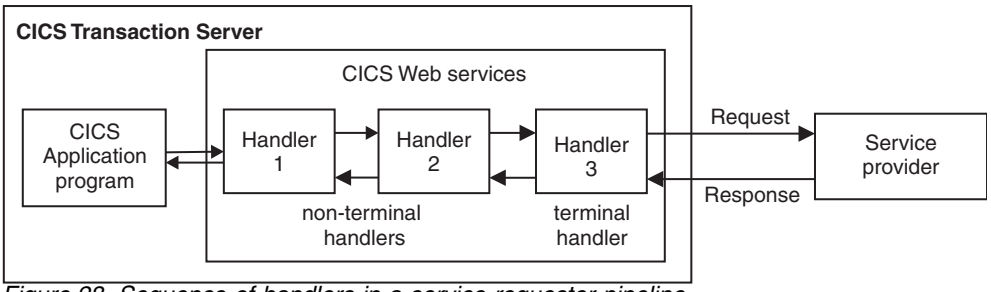


Figure 28. Sequence of handlers in a service requester pipeline

Sequence	Handler	DFHFUNCTION
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 DFHFUNCTION that can be encountered in a given message handler depend on 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 nonterminal handler. The following table summarizes when each value can occur:

Value of DFHFUNCTION	Provider or requester pipeline	Pipeline phase	Terminal or nonterminal handler
RECEIVE-REQUEST	Provider	Request phase	Nonterminal
SEND-RESPONSE	Provider	Response phase	Nonterminal
SEND-REQUEST	Requester	Request phase	Nonterminal
RECEIVE-RESPONSE	Requester	Response phase	Nonterminal
PROCESS-REQUEST	Provider	Request phase	Terminal
NO-RESPONSE	Both	Response phase	Nonterminal
HANDLER-ERROR	Both	Both	Both

DFHHTTPMETHOD container

This is a container of DATATYPE(CHAR) that is available to application programs in all HTTP provider mode CICS pipelines.

This container is 8 characters long and holds the name of the HTTP method that was used on the incoming request. It is not populated if the request did not arrive over HTTP.

DFHHTTPSTATUS container

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.

tttttttt
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.

DFHMEDIATYPE container

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.

DFHNORESPONSE container

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 the DFHNORESPONSE container 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, no response is expected.
- If container DFHNORESPONSE is absent, a response *is* expected.

This information is provided, initially, by the service requester application, based on the protocol used with the service provider. Therefore, you are advised not to delete this container in a message handler (or to create it, if it does not exist), because doing so might disturb the protocol between the endpoints.

Other than in the request phase of a service requester pipeline, the use of this container is not defined.

DFHREQUEST container

DFHREQUEST is a container of DATATYPE(CHAR) that contains the request message that is processed in the request phase of a pipeline.

If one of the CICS-supplied SOAP message handlers is configured in the pipeline, the container DFHREQUEST is updated to include the SOAP message headers in the SOAP envelope. If the message is constructed by a CICS-supplied SOAP message handler, and has not been changed subsequently, DFHREQUEST contains a complete SOAP envelope and all of its contents is in the UTF-8 code page.

The DFHREQUEST container is present in the request when a message handler is called, and the DFHFUNCTION container 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 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 the DFHRESPONSE container. In this way, the normal sequence of processing is reversed, and the processing continues with the response phase of the pipeline.

DFHRESPONSE container

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.

The DFHRESPONSE container is present when a message handler is called, and the DFHFUNCTION container 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.

The DFHRESPONSE container is also present, with a length of zero, when DFHFUNCTION contains `RECEIVE-REQUEST`, `SEND-REQUEST`, `PROCESS-REQUEST`, or `HANDLER-ERROR`.

DFHWS-CCSID container

DFHWS-CCSID is a container of DATATYPE(BIT) that contains a fullword (4 bytes) specifying the CCSID of the data in the response container.

The container is valid only for a provider mode pipeline that uses CICS code to transform the language structure into XML.

The CCSID must be compatible with the CCSID used to generate the WSBIND file. If it is not, the SOAP response that is produced might contain incorrect or invalid characters.

The CCSID is not allowed to be changed to or from 930, 1390, 5026 and 1026. Also CICS does not allow the CCSID to be changed to one that is usable as a client CCSID.

If there are any problems processing the value in the DFHWS-CCSID container, processing continues using the CCSID from the WSBIND file.

The DFHWS-CCSID container is checked only on return from a channel driven application program.

How containers control the pipeline protocols

The contents of the DFHFUNCTION, DFHREQUEST, and DFHRESPONSE containers together control the pipeline protocols.

During the two phases of the execution of a pipeline (the request phase and the response phase) the value of DFHFUNCTION determines which control containers are passed to each message handler:

DFHFUNCTION	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 the containers that your message handler passes back to the pipeline:

During the request phase

- Your message handler can return the DFHREQUEST container. 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 the DFHRESPONSE container. Processing switches to the response phase, and the same handler is called with DFHFUNCTION set to SEND-RESPONSE in a service provider and to 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 called with DFHFUNCTION set to NO-RESPONSE.

In the terminal handler (service provider only)

- Your message handler can return the DFHRESPONSE container. Processing switches to the response phase, and the previous handler is called with a new value of DFHFUNCTION (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 called with a new value of DFHFUNCTION (NO-RESPONSE).

During the response phase

- Your message handler can return the DFHRESPONSE container. Processing continues in the response phase, and the next handler is called. 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 called with a new value of DFHFUNCTION (NO-RESPONSE).

Important: During the request phase, your message handler can return DFHREQUEST or DFHRESPONSE, but not both. Because both containers are present when your message handler is called, you must delete one of them. This table shows the action taken by the pipeline for all values of DFHFUNCTION 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)
RECEIVE-REQUEST	Service provider; request phase	Present (length > 0)	Absent	Call the next handler with the RECEIVE-REQUEST function
RECEIVE-REQUEST	Service provider; request phase	Present (length = 0)	Not applicable	(error)
RECEIVE-REQUEST	Service provider; request phase	Absent	Present (length > 0)	Switch to response phase, and invoke the same handler with the SEND-RESPONSE function
RECEIVE-REQUEST	Service provider; request phase	Absent	Present (length = 0)	(error)
RECEIVE-REQUEST	Service provider; request phase	Absent	Absent	Call the same handler with the NO-RESPONSE function

DFHFUNCTION	Context	DFHREQUEST	DFHRESPONSE	Action
SEND-RESPONSE	Service provider; response phase	Not applicable	Present (length > 0)	Call the previous handler with the SEND-RESPONSE function
SEND-RESPONSE	Service provider; response phase	Not applicable	Present (length = 0)	(error)
SEND-RESPONSE	Service provider; response phase	Not applicable	Absent	Call the same handler with the NO-RESPONSE function
SEND-REQUEST	Service requester; request phase	Present (length > 0)	Present (length ≥ 0)	(error)
SEND-REQUEST	Service requester; request phase	Present (length > 0)	Absent	Call the next handler with the SEND-REQUEST function
SEND-REQUEST	Service requester; request phase	Present (length = 0)	Not applicable	(error)
SEND-REQUEST	Service requester; request phase	Absent	Present (length > 0)	Switch to response phase, and call the previous handler with the RECEIVE-RESPONSE function
SEND-REQUEST	Service requester; request phase	Absent	Present (length = 0)	(error)
SEND-REQUEST	Service requester; request phase	Absent	Absent	Call the same handler with the NO-RESPONSE function
RECEIVE-RESPONSE	Service requester; response phase	Not applicable	Present (length > 0)	Call the previous handler with the RECEIVE-RESPONSE function
RECEIVE-RESPONSE	Service requester; response phase	Not applicable	Present (length = 0)	(error)
RECEIVE-RESPONSE	Service requester; response phase	Not applicable	Absent	Call the same handler with the NO-RESPONSE function
PROCESS-REQUEST	Service provider; terminal handler	Not applicable	Present (length > 0)	Call the previous handler with the RECEIVE-RESPONSE function
PROCESS-REQUEST	Service provider; terminal handler	Not applicable	Present (length = 0)	(error)
PROCESS-REQUEST	Service provider; terminal handler	Not applicable	Absent	Call the same handler with the NO-RESPONSE function
HANDLER-ERROR	Service requester or provider; either phase	Not applicable	Present (length > 0)	Call the previous handler with the SEND-RESPONSE function or the RECEIVE-RESPONSE function
HANDLER-ERROR	Service requester or provider; either phase	Not applicable	Present (length = 0)	(error)
HANDLER-ERROR	Service requester or provider; either phase	Not applicable	Absent	Call the same handler with the NO-RESPONSE function

Context containers

In some situations, user-written message handler programs, and header processing programs, need information about the context in which they are called. 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 user ID 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.

DFH-EXIT-HEADER1 container

DFH-EXIT-HEADER1 is a container of DATATYPE(CHAR). It contains one or more SOAP headers that are added to a response from a web service provider application in CICS.

Programs running global user exit XWSPRRWO can add a header to a SOAP response. The header must be valid SOAP and the name spaces must be self-contained in the header XML. A program that puts data in this container must check for its presence and add the new header to the end of the data. By following this best practice, multiple programs can be driven at the same exit point if required.

DFH-HANDLERPLIST container

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, the container is empty; that is, it has a length of zero.

You cannot change the contents of this container.

DFH-SERVICEPLIST container

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, the container is empty; that is, it has a length of zero.

You cannot change the contents of this container.

DFHWS-APPHANDLER container

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 a pipeline that contains the `<apphandler>` element, the 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.

Related concepts:

"Application handlers" on page 275
An application handler is a CICS program that the terminal handler of a SOAP

service provider pipeline links to at run time.

Related reference:

"The <apphandler> element" on page 236
Specifies the name of the application handler that the terminal handler of the pipeline links to by default.

DFHWS-APPHANCLAS container

DFHWS-APPHANCLAS is a container of DATATYPE(CHAR) that, in a service provider pipeline, is initialized with the contents of the <apphandler_class> element of the pipeline configuration file.

In the terminal handler of a Java-based pipeline, the supplied SOAP handlers, <cics_soap_1.1_handler_java> and <cics_soap_1.2_handler_java>, get the name of the target application program from this container.

CICS does not provide this container in a service requester pipeline.

Related concepts:

"Application handlers" on page 275
An application handler is a CICS program that the terminal handler of a SOAP service provider pipeline links to at run time.

Related reference:

"The <apphandler_class> element" on page 236
Specifies that the terminal handler of the pipeline links to an Axis2 application handler.

DFHWS-DATA container

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, the DFHWS-DATA container must be present when the service requester program issues an **EXEC CICS INVOKE SERVICE** 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, the DFHWS-DATA container is used by default when you do not specify the **CONTID** parameter on the DFHLS2WS 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.

DFHWS-FAULT container

DFHWS-FAULT is a container of DATATYPE(BIT) that holds information about the type of SOAP fault CICS generates.

The container holds a binary fullword that indicates the fault type that can be used in further processing for a web service response:

1. The most recent SOAP fault was for a CICS fault (for example, CICS or user abend).

2. The most recent SOAP fault was for an application fault. The container is deleted when you issue the EXEC CICS SOAPFAULT DELETE command. If a second or new SOAP fault is created, CICS updates the new container appropriately.

You cannot change the contents of this container.

DFHWS-LOCATION container

DFHWS-LOCATION is a container of DATATYPE(CHAR) that contains supplied Location header when the HTTP response was 302, 303 or 307.

DFHWS-MEP container

DFHWS-MEP is a container of DATATYPE(BIT) that holds a representative value for the message exchange pattern (MEP) of an inbound or outbound SOAP message. This value is one byte in length.

CICS supports four message exchange patterns for both service requesters and service providers. The message exchange pattern is defined in the WSDL 2.0 document for the web service and determines whether CICS responds as the provider, and if CICS expects a response from an external provider. In requester mode, the time that CICS waits for a response is configured using the PIPELINE resource.

If you used the CICS web services assistant to deploy your application, this container is populated by CICS:

- In a service provider pipeline, this container is populated by the DFHPITP application handler when it receives the inbound message from the terminal handler.
- In a service requester pipeline, this container is populated when the application uses the **INVOKE SERVICE** command.

If the application uses the DFHPIRT channel to start the pipeline, the application populates this container. If the container is not present or has no value, CICS assumes that the request is using either the In-Out or In-Only MEP, depending on whether the DFHNORESPONSE container is present in the channel.

This container is populated by the supplied application handler program, DFHPITP. If you use a different application handler then this container is not available for use.

Table 9. Values that can appear in container DFHWS-MEP

Value	MEP	URI
1	In-Only	http://www.w3.org/ns/wsdl/in-only
2	In-Out	http://www.w3.org/ns/wsdl/in-out
4	Robust-In-Only	http://www.w3.org/ns/wsdl/robust-in-only
8	In-Optional-Out	http://www.w3.org/ns/wsdl/in-opt-out

DFHWS-OPERATION container

DFHWS-OPERATION is a container of DATATYPE(CHAR) that is usually 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 called. It is populated when a supplied SOAP message

handler passes control to the target application program, and is visible only when the target program is called with a channel interface.

In a service requester pipeline, the container holds the name specified in the OPERATION option of the **EXEC CICS INVOKE SERVICE** command. The container is not available to the application that issues the command.

This container is populated by the supplied application handler program, DFHPITP. If you use a different application handler then this container is not available for use.

DFHWS-PIPELINE container

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.

DFHWS-RESPWAIT container

DFHWS-RESPWAIT is a container of DATATYPE(BIT) that contains an unsigned fullword binary number to represent the timeout in seconds that applies to outbound web service request and response messages.

The value of this container is supplied by the RESPWAIT attribute of the PIPELINE definition and is set by CICS when the INVOKE SERVICE command is issued. Any value set in this container by the user application before the INVOKE SERVICE command is issued will be ignored.

A message handler program that is invoked during pipeline processing can overwrite the value of the DFHWS-RESPWAIT container. If this is done, the updated value is only used if the PIPELINE definition has a RESPWAIT attribute that is not set to DEFT or left blank. If the PIPELINE definition has the RESPWAIT attribute set to DEFT or left blank, the default timeout value of the transport protocol is always used, regardless of the value in the DFHWS-RESPWAIT container.

This container is used only in requester mode pipelines.

DFHWS-SOAPLEVEL container

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 holds a binary fullword that indicates the level of SOAP that 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.

DFHWS-TRANID container

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 runs in a new task with the new transaction ID.

New tasks cannot be started when both the terminal handler and the application handler of a pipeline run in the same JVM server. For this reason, if you deploy JAX-WS Axis2 applications into CICS, DFHWS-TRANID cannot be used to change the user ID.

DFHWS-URI container

DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

In a service provider pipeline, CICS extracts the relative URI from the incoming message and places it in the DFHWS-URI container.

For example, if the URI of the web services is `http://example.com/location/address` or `jms://queue?destination=INPUT.QUEUE&targetService=/location/address`, the relative URI is `/location/address`.

If you are using Web Services Addressing in your requester pipeline, this container will be created and updated in the following order:

1. When the INVOKE SERVICE command runs, it creates the DFHWS-URI container and initiates it with the value of the WSDL service endpoint address. If the WSACONTEXT BUILD API command was used to create an addressing context, you must not specify the URI or URIMAP parameters on the INVOKE SERVICE command.
2. When the web services addressing handler (DFHWSADH) runs, if a <wsa:To> EPR exists in the addressing context with a non-anonymous URI, the URI in the DFHWS-URI container is overwritten with the value of the <wsa:To> EPR. The anonymous URI is ignored.

The SOAP message is sent to the service defined by the URI in DFHWS-URI.

In a service requester pipeline, CICS puts the URI that is specified on the **INVOKE SERVICE** command, or, if missing, the URI from the web service binding, in the DFHWS-URI container. You can override this URI by using a message handler in the pipeline.

A service can use an HTTP, HTTPS, JMS, or WebSphere MQ URI for external services. A service can also use a CICS URI for a service that is provided by another CICS application:

URI	Query string	Description
`cics://PROGRAM/`*program*	`?`*options*	The CICS transport handler uses an **EXEC CICS LINK PROGRAM** command to link to the specified program, passing the current channel and containers. No data transformation takes place on the application data.

URI	Query string	Description
cics://SERVICE/*service*	?targetServiceUri=*targetServiceUri*&*options*	The CICS transport handler uses the path of the service, expressed as the *targetServiceUri*, to match a URIMAP resource to run the request through a provider pipeline. You must specify a value for the **targetServiceUri** parameter if you use this URI type.
cics://PIPELINE/*pipeline*	?targetServiceUri=*targetServiceUri*	The CICS transport handler starts another service requester pipeline.

You can add parameters to each type of CICS URI using the format *parameter=value*, where each parameter is separated by an ampersand. The following rules apply to the CICS URI:

- The first parameter in the query string must be prefixed with a question mark. You cannot use a question mark before this point in the URI.
- To include an ampersand in a parameter value, you must escape the character. For more information, see the example section at the end of this topic.
- CICS changes any lowercase values for *program* and *pipeline* to uppercase.

The parameters on the query string determine how CICS processes the request at the end of the requester pipeline:

maxCommareaLength=*value*
: Specify the maximum size of the COMMAREA in bytes, that is required for the target application program. The value must not exceed 32 763. If this parameter is present in the query string, CICS links to the specified program using a COMMAREA. If this parameter is not present in the query string, CICS links to the specified program using a channel.

  This parameter not case-sensitive and is valid only for the cics://PROGRAM URI.

newTask=*yes*|no
: Specify whether the transport handler will run the request as a new task.

  This parameter is not case-sensitive. cics://PROGRAM/testapp?newTask=yes and cics://PROGRAM/testapp?NEWTASK=Yes are the same.

targetServiceUri=*uri*
: Specify the path of the service to be called. On a SERVICE destination type, the transport handler uses the value with host=localhost to locate the URIMAP resource to start a service provider pipeline. On a PIPELINE destination type, the transport handler uses the value to start another requester pipeline.

  This parameter is case-sensitive.

transid=*char(4)*
: Specify a transaction under which the request will run. The transport handler starts a request stream using the specified transaction ID.

  This parameter is case-sensitive.

userid=*char(8)*
: Specify a user ID under which the request will run. The transport handler starts a request stream using the specified user ID.

This parameter is not case-sensitive.

Destination type	Parameters on URI	
PROGRAM	**userid**	Optional
PROGRAM	**transid**	Optional
PROGRAM	**maxCommareaLength**	Optional
PROGRAM	**newTask**	Optional. Must be yes or not specified at all if you specify **userid** or **transid**.
PROGRAM	**targetServiceUri**	Not supported
SERVICE	**userid**	Optional
SERVICE	**transid**	Optional
SERVICE	**maxCommareaLength**	Not supported
SERVICE	**newTask**	Optional. Must be yes or not specified at all if you specify **userid** or **transid**.
SERVICE	**targetServiceUri**	Required
PIPELINE	**userid**	Not supported
PIPELINE	**transid**	Not supported
PIPELINE	**maxCommareaLength**	Not supported
PIPELINE	**newTask**	Not supported
PIPELINE	**targetServiceUri**	Required

Examples of CICS URIs

In this first example, the DFHWS-URI container has the following URI by the time it reaches the end of the pipeline:

```
cics://PROGRAM/testapp?newTask=yes&userid=user1
```

The transport handler links to the CICS program called testapp, passing the channel and containers. No data transformation takes place, so the target program must be able to process the contents of the containers on the current channel. CICS links to the program under a new unit of work and a different user ID of user1.

In this second example, the DFHWS-URI container has the following URI by the time it reaches the end of the pipeline:

```
cics://SERVICE/getStockQuote?targetServiceUri=/stock/getQuote&newTask=yes&userid=user2
```

The transport handler replaces the URI in the DFHWS-URI container with the value `/stock/getQuote`, finds the URIMAP using the path in the **targetServiceUri** parameter to resolve the URI, and starts the provider pipeline under a new task and different user ID.

In this third example, the DFHWS-URI container has the following URI by the time it reaches the end of the pipeline:

```
cics://PIPELINE/reqpipeA?targetServiceUri=cics://PROGRAM/testapp?newTask=yes%26userid=user1
```

The transport handler replaces the URI in the DFHWS-URI container with the value `cics://PROGRAM/testapp?newTask=yes&userid=user1` and starts the requester pipeline called reqpipeA, passing the current channel and containers. The `%26`

characters escape the ampersand, so the transport handler puts the whole URI in the DFHWS-URI container.

Related concepts:

"Options for controlling requester pipeline processing" on page 325
In service requester pipelines, message handlers can determine where the web service request is sent by changing the URI. CICS provides support for different URI formats so that you have much more flexibility in the way that the pipeline processes web service requests.

Related tasks:

"Controlling requester pipeline processing using a URI" on page 327
In service requester pipelines, a message handler can determine where to send the web service request by changing the URI. By changing the URI format, you can choose to perform certain optimizations, such as starting another requester pipeline or starting a service provider pipeline without sending the request over the network.

DFHWS-URI-RESID container

This is a container of DATATYPE(CHAR) that is only available to applications attached by a JSON pipeline.

This container holds a simplified copy of the URI path (a RESource IDentifier), in which the path URI fragment that was used for URIMAP matching has been removed. For example,

If the URIMAP that matched the incoming request has a PATH of:

```
/JSONServices/CustomerDetails/*
```

and the incoming URI from the client was:

```
http://www.example.org:10000/JSONServices/CustomerDetails/customerNumber/13388?action=query
```

then the contents of DFHWS-URI-RESID would be:

```
customerNumber/13388
```

RESTful JSON applications will be able to use this container to help identify the resource id (or primary key) for RESTful resources that are matched using a wild-carded URIMAP. This should be significantly simpler than parsing through the contents of DFHWS-URI.

Note: If the PATH attribute of the matching URIMAP isn't wild-carded (i.e. it contained the complete Path for the URI), the contents of this Container will be empty.

Note: The PATH attribute of the matching URIMAP may contain an optional query string fragment. If so, the query string fragment is ignored when constructing this container.

DFHWS-URI-QUERY container

This is a container of DATATYPE(CHAR), that is available to application programs in all HTTP provider mode CICS pipelines.

This container holds the query string fragment of the URI. For example,

If the incoming URI from the client was:

```
http://www.example.org:10000/JSONServices/CustomerDetails/customerNumber/13388?action=query&page=1
```

then the contents of DFHWS-URI-QUERY would be:

```
action=query&page=1
```

Applications may parse through the contents of this Container to identify individual **name=value** parameters from the URI.

Note: If the incoming URI did not include a query string then this Container will not be present on the Channel.

DFHWS-URIMAPPATH container

This is a container of DATATYPE(CHAR), and holds a copy of the PATH data from the URIMAP that was used to match the incoming URI.

Any pipeline attached application may make use of this Container to understand more about how it came to be attached.

DFHWS-USERID container

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 runs in a new task that is associated with the new user ID. Unless you change the contents of container DFHWS-TRANID, the new task has the same transaction ID as the task in which the pipeline is running.

New tasks cannot be started when both the terminal handler and the application handler of a pipeline run in the same JVM server. For this reason, if you deploy JAX-WS Axis2 applications into CICS, DFHWS-USERID cannot be used to change the user ID.

DFHWS-WEBSERVICE container

DFHWS-WEBSERVICE is a container of DATATYPE(CHAR) that is used in a service provider pipeline only. It holds the name of the web service 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.

DFHWS-CID-DOMAIN container

DFHWS-CID-DOMAIN is a container of DATATYPE(CHAR). It contains the domain name that is used to generate content-ID values for referencing binary attachments.

The value of the domain name is `cicsts` by default. You can override the value by specifying the `<mime_options>` element in the pipeline configuration file.

You cannot change the contents of this container.

DFHWS-MTOM-IN container

DFHWS-MTOM-IN is a container of DATATYPE(BIT) that holds information about the specified options for the `<cics_mtom_handler>` element of the pipeline configuration file and information about the message format that has been received in the pipeline.

It contains the information to process an inbound MTOM message in the pipeline. The inbound message can be a request message from a web service requester or a response message from a web service provider.

If you do not specify a <cics_mtom_handler> element in the pipeline configuration file, or if a SOAP message is received instead of an MTOM message, this container is not created.

If web services security is configured in the pipeline, or if validation is switched on for a web service, the contents of field XOP_MODE in DFHWS-MTOM-IN can be overridden by CICS when the container is created. For example, if you configure the pipeline to process the content of MTOM messages in direct mode, and you then switch validation on for the web service, CICS overrides the defined value in the pipeline configuration file and sets the XOP processing to run in compatibility mode. CICS performs the override because of the restrictions in support for processing XOP documents and binary attachments in the pipeline.

You cannot change the contents of this container.

Table 10. Structure of the DFHWS-MTOM-IN container

Field name	Length (bytes)	Contents
MTOM_STATUS	4	Contains the value "1", indicating that the message received by CICS is in MTOM format.
MTOMNOXOP_STATUS	4	Contains one of the following values: **0** The MTOM message contains binary attachments. **1** The MTOM message does not contain binary attachments.
XOP_MODE	4	Contains one of the following values: **0** No XOP processing takes place. **1** XOP processing takes place in compatibility mode. **2** XOP processing takes place in direct mode.

DFHWS-MTOM-OUT container

DFHWS-MTOM-OUT is a container of DATATYPE(BIT) that holds information about the specified options for the <cics_mtom_handler> element of the pipeline configuration file.

It contains the information to process an outbound MTOM message in the pipeline, whether it is a response message for a web service requester or a request message for a web service provider.

If you do not specify a <cics_mtom_handler> element in the pipeline configuration file, or if the <mtom_options> element in the pipeline configuration file has the attribute send_mtom="no", this container is not created.

In provider mode, this container is created at the same time as the DFHWS-MTOM-IN container. If the <mtom_options> element in the pipeline configuration file has the attribute send_mtom="same", the MTOM_STATUS field is set to indicate whether the web service requester wants an MTOM or SOAP response message.

If web services security is configured in the pipeline, or if validation is switched on for a web service, the XOP_MODE field of DFHWS-MTOM-OUT can be changed by CICS when the container is created. For example, if you configure the pipeline to process the XOP document and any binary attachments using direct mode, and you then switch validation on for a web service, CICS overrides the defined value in the pipeline configuration file and sets the XOP processing to run in compatibility mode when it creates the container. CICS performs the override because of restrictions in support for processing XOP documents and binary attachments in the pipeline.

You cannot change the contents of this container.

Table 11. Structure of the DFHWS-MTOM-OUT container

Field name	Length (bytes)	Contents
MTOM_STATUS	4	Indicates whether MTOM is enabled: **0** MTOM is not enabled. The outbound message is sent in SOAP format. **1** MTOM is enabled. The outbound message is sent in MTOM format.
MTOMNOXOP_STATUS	4	Indicates whether to use MTOM when there are no binary attachments: **0** Do not send an MTOM message when there are no binary attachments. **1** Send an MTOM message when there are no binary attachments.
XOP_MODE	4	Indicates what XOP processing should take place: **0** No XOP processing takes place. **1** XOP processing takes place in compatibility mode. **2** XOP processing takes place in direct mode.

DFHWS-WSDL-CTX container

DFHWS-WSDL-CTX is a container of DATATYPE(CHAR) that is used in either a service provider or a service requester application deployed with the CICS web services assistant. It holds WSDL context information that can be used for monitoring purposes.

DFHWS-WSDL-CTX holds the following context information for the WSDL document:

- The name and namespace of the operation for which the application is being invoked.
- If known, the name and namespace for the WSDL 1.1 port or WSDL 2.0 endpoint that is being used.

These values are separated by space characters. DFHWS-WSDL-CTX is populated by CICS only at runtime level 2.1 and later.

If you used the CICS web services assistant to deploy your application, this container is populated by CICS:

- In a service provider pipeline, this container is populated by the DFHPITP application handler when it receives the inbound message from the terminal handler.
- In a service requester pipeline, this container is populated when the application uses the **INVOKE SERVICE** command.

If the application uses the DFHPIRT program to start the pipeline, the application populates the DFHWS-WSDL-CTX container if required.

DFHWS-XOP-IN container

DFHWS-XOP-IN is a container of DATATYPE(BIT). It contains a list of references to the binary attachments that have been unpackaged from an inbound MIME message and placed in containers using XOP processing.

Each attachment record in the DFHWS-XOP-IN container consists of these items:

- The 16-byte name of the container that holds the MIME headers for the binary attachment
- The 16-byte name of the container that holds the binary attachment
- The 2-byte length of the content-ID, in signed halfword binary format
- The content-ID, including the < and > delimiters, stored as an ASCII character string

You cannot change the contents of this container.

DFHWS-XOP-OUT container

DFHWS-XOP-OUT is a container of DATATYPE(BIT). It contains a list of references to the containers that hold binary attachments. The binary attachments are packaged into an outbound MIME message by the MTOM handler program.

Each attachment record in the DFHWS-XOP-OUT container consists of these items:

- The 16-byte name of the container that holds the MIME headers for the binary attachment
- The 16-byte name of the container that holds the binary attachment
- The 2-byte length of the content-ID, in signed halfword binary format
- The content-ID, including the < and > delimiters, stored as an ASCII character string

You cannot change the contents of this container.

The header processing program containers

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, and DFHXMLSS-PARSE.

Note: Although web service that use Axis2 to process SOAP messages can use the header processing program interface, it is more efficient to write your own Axis2

handlers in Java to process the SOAP headers. For more information on creating Axis2 handlers, see Writing Your Own Axis2 Module

Container DFHHEADER

When the header processing program is called, DFHHEADER contains the single header block that 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 called even when there is no matching header block in the SOAP message. In this case, container DFHHEADER has a length of zero. This is the case when a header processing program is called 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 called, by specifying `<mandatory>true</mandatory>` or `<mandatory>1</mandatory>`.

When the header program returns, container DFHHEADER must contain zero, one, or more header blocks that 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 called, DFHWS-XMLNS contains information about XML namespaces that are declared in the SOAP envelope. The header program can use this information to perform the following tasks:

- Resolve qualified names that it encounters in the header block
- 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 that are not visible in the program may rely on them.

Container DFHWS-BODY

This container contains the body section of the SOAP envelope. The header processing program can modify the contents.

When the header processing program is called, 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.

Container DFHXMLSS-PARSE

When you use either the <cics_soap_1.1_handler> or <cics_soap_1.2_handler> elements in your pipeline configuration, and header program is called, DFHXMLSS-PARSE contains the XML System Services (XMLSS) records for that header. This container is not created when <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> elements are used.

Control, context, and user containers

As well as the containers described, the interface passes the *control containers*, *context containers*, and *user containers* on channel DFHHHC-V1.

For more information about these containers, see "Containers used in the pipeline" on page 290.

Security containers

Security containers are used on the DFHWSTC-V1 channel to send and receive identity tokens from a Security Token Service (STS) such as Tivoli Federated Identity Manager. This interface is called the *Trust client interface* and can be used in web service requester and provider pipelines.

DFHWS-IDTOKEN container

DFHWS-IDTOKEN is a container of DATATYPE(CHAR). It contains the token that the Security Token Service (STS) either validates or uses to issue an identity token for the message.

The token must be in XML format.

Use this container only with channel DFHWSTC-V1 for the Trust client interface.

DFHWS-RESTOKEN container

DFHWS-RESTOKEN is a container of DATATYPE(CHAR). It contains the response from the Security Token Service (STS).

The response depends on the action that was requested from the STS in the DFHWS-STSACTION container.

- If the action is issue, this container holds the token that the STS has exchanged for the one that was sent in the DFHWS-IDTOKEN container.
- If the action is validate, this container holds a URI to indicate whether the security token that was sent in the DFHWS-IDTOKEN container is valid or not valid. The URIs that can be returned are as follows:

URI	Description
http://schemas.xmlsoap.org/ws/2005/02/trust/status/valid	The security token is valid.
http://schemas.xmlsoap.org/ws/2005/02/trust/status/invalid	The security token is not valid.

This container is returned on the channel DFHWSTC-V1 when using the Trust client interface.

DFHWS-SERVICEURI container

DFHWS-SERVICEURI is a container of DATATYPE(CHAR). It contains the URI that the Security Token Service (STS) uses as the AppliesTo scope.

The AppliesTo scope is used to determine the web service with which the security token is associated.

Use this container only with channel DFHWSTC-V1 for the Trust client interface.

DFHWS-STSACTION container

DFHWS-STSACTION is a container of DATATYPE(CHAR). It contains the URI of the action that the Security Token Service (STS) takes to either validate or issue a security token.

The URI values that you can specify in this container are as follows:

URI	Description
http://schemas.xmlsoap.org/ws/2005/02/trust/Issue	The STS issues a token in exchange for the one that is sent in the DFHWS-IDTOKEN container.
http://schemas.xmlsoap.org/ws/2005/02/trust/Validate	The STS validates the token that is sent in the DFHWS-IDTOKEN container.

Use this container only with channel DFHWSTC-V1 for the Trust client interface.

DFHWS-STSFAULT container

DFHWS-STSFAULT is a container of DATATYPE(CHAR). It contains the error that was returned by the Security Token Service (STS).

If an error occurs, the STS issues a SOAP fault. The contents of the SOAP fault are returned in this container.

This container is returned on the channel DFHWSTC-V1 when using the Trust client interface.

DFHWS-STSREASON container

DFHWS-STSREASON is a container of DATATYPE(CHAR). It contains the contents of the <wst:Reason> element, if this element is present in the response message from the Security Token Service (STS).

The <wst:Reason> element contains an optional string that provides information relating to the status of the validation request that was sent to the STS by CICS. If the security token is not valid, the information provided by the STS in this element can help you to determine why the token is not valid.

For more information, see the *Web Services Trust Language* specification that is published at OASIS WS-Trust v1.4 Standard.

DFHWS-STSURI container

DFHWS-STSURI is a container of DATATYPE(CHAR). It contains the absolute URI of the Security Token Service (STS) that is used to validate or issue an identity token for the SOAP message.

The format of the URI is `http://www.example.com:8080/TrustServer/SecurityTokenService`. You can use HTTP or HTTPS, depending on your security requirements.

Use this container only with channel DFHWSTC-V1 for the Trust client interface.

DFHWS-TOKENTYPE container

DFHWS-TOKENTYPE is a container of DATATYPE(CHAR). It contains the URI of the requested token type that the Security Token Service (STS) issues as an identity token for the SOAP message.

You can specify any valid token type, but it must be supported by the STS.

Use this container only with channel DFHWSTC-V1 for the Trust client interface.

SAML support containers

The read-only containers that are used by CICS SAML support.

In the following topics, *nnn* means that there might be more than one container. Containers are numbered 001 to *nnn* (the number of containers of this type returned). More than 999 containers of a particular type are not supported and the data in the SAML assertion that relates to them is ignored. Containers that are not mapped to a DSECT are variable length.

DFHSAML-AnnnVmmm container

DFHSAML-AnnnVmmm is a container of DATATYPE(CHAR). It contains Attribute Value *mmm* for attribute *nnn*, where *nnn* and *mmm* are 3-digit numbers.

The number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.

The number of values for this attribute is in DFHSAML-ATTRAnnn.

DFHSAML-ASSQNAME container

DFHSAML-ASSQNAME is a container of DATATYPE(CHAR). It contains the SAML Assertion namespace.

Possible values are

SAML 1.1
urn:oasis:names:tc:SAML:1.0:assertion

SAML 2.0
urn:oasis:names:tc:SAML:2.0:assertion

This assertion must be a URI. If the assertion is more complex, it extracts into the 3 parts.

DFHSAML-ATTRAnnn container

DFHSAML-ATTRAnnn is a container of DATATYPE(BIT). It contains a BIN(31) field with the number of values for attribute *nnn*. The maximum number of values is 999.

The number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.

DFHSAML-ATTRFnnn container

DFHSAML-ATTRFnnn is a container of DATATYPE(CHAR). It contains the Attribute name format for attribute *nnn*, where *nnn* is a 3-digit number.

The number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.

DFHSAML-ATTRNnnn container

DFHSAML-ATTRNnnn is a container of DATATYPE(CHAR). It contains the Attribute Name for attribute *nnn*, where *nnn* is a 3-digit number.

The number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.

DFHSAML-ATTRSnnn container

DFHSAML-ATTRSnnn is a container of DATATYPE(CHAR). It contains the Attribute Name Space for attribute *nnn*, where *nnn* is a 3-digit number.

The number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.

DFHSAML-ATTRYnnn container

DFHSAML-ATTRYnnn is a container of DATATYPE(CHAR). It contains the Attribute Friendly name for attribute *nnn*, where *nnn* is a 3-digit number.

The number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.

DFHSAML-AUDNRnnn container

DFHSAML-AUDNRnnn is a container of DATATYPE(CHAR). It contains the AudienceRestriction name.

The number of containers returned is AUDNRNUM.

DFHSAML-AUTHMETH container

DFHSAML-AUTHMETH is a container of DATATYPE(CHAR). It contains the method that is used to authenticate the token holder.

Methods include `password`, `Kerberos`, and `ltpa`.

DFHSAML-CERTIDN container

DFHSAML-CERTIDN is a container of DATATYPE(CHAR). It contains the issuer's distinguished name of the SAML signer's X.509 Certificate.

DFHSAML-CERTSDN container

DFHSAML-CERTSDN is a container of DATATYPE(CHAR). It contains the subject's distinguished name of the SAML signer's X.509 Certificate.

DFHSAML-CERTSNUM container

DFHSAML-CERTSNUM is a container of DATATYPE(CHAR). It is an eight-character field that contains the SAML signer's X.509 Certificate serial number.

DFHSAML-CONFMETH container

DFHSAML-CONFMETH is a container of DATATYPE(CHAR). It contains the SubjectConfirmation method that is used in this SAML token.

Valid methods are holder-of-key, bearer, or sender-vouches. The returned string is based on the OASIS SAML token profile 1.1 and 2.0.

Note: SAML tokens that have more than one confirmation method are not supported. If there is more than one confirmation method, the results are unpredictable.

DFHSAML-COUNTS container

DFHSAML-COUNTS is a container of DATATYPE(BIT). It contains the number of variable length containers output.

DFHSAML-FLAGS container

DFHSAML-FLAGS is a container of DATATYPE(CHAR). It contains a collection of flag bytes.

DFHSAML-ISSUER container

DFHSAML-ISSUER is a container of DATATYPE(CHAR). It contains the name of the issuer.

DFHSAML-NAMID container

DFHSAML-NAMID is a container of DATATYPE(CHAR). It contains the value of the name format property.

DFHSAML-NAMIDF container

DFHSAML-NAMIDF is a container of DATATYPE(CHAR). It contains a URI reference that represents the classification of string-based identifier information.

DFHSAML-NAMIDQ container

DFHSAML-NAMIDQ is a container of DATATYPE(CHAR). It contains the security or administrative domain that qualifies the name.

DFHSAML-NAMIDSP container

DFHSAML-NAMIDSP is a container of DATATYPE(CHAR). It contains the name identifier that is established by a service provider or affiliation of providers of the entity.

DFHSAML-NAMIDSPQ container

DFHSAML-NAMIDSPQ is a container of DATATYPE(CHAR). It contains the name of a service provider or affiliation of providers.

DFHSAML-OUTTOKEN container

DFHSAML-OUTTOKEN is a container of DATATYPE(CHAR). It contains a SAML token.

If this is an input container, it contains the previously validated token, which is being routed to another service provider, or extended and then routed.

If this is an output container, it contains a SAML token output by DFHSAML processing. If the processing is validation or extraction, this token is the validated,extracted, or modified and resigned token.

DFHSAML-PROXYnnn container

DFHSAML-PROXYnnn is a container of DATATYPE(CHAR). It contains the ProxyRestriction Audience name.

DFHSAML-RESPONSE container

DFHSAML-RESPONSE is a container of DATATYPE(BIT). It contains a response code that is used internally.

DFHSAML-SAMLID container

DFHSAML-SAMLID is a container of DATATYPE(CHAR). It contains a string that represents the ID for SAML 2.0, or AssertionID for SAML 1.1.

DFHSAML-SUBJADDR container

DFHSAML-SUBJADDR is a container of DATATYPE(CHAR). It contains the IP address in SubjectLocality.

Restriction: This container is not returned for SAML 2.0.

DFHSAML-SUBJDNS container

DFHSAML-SUBJDNS is a container of DATATYPE(CHAR). It contains the DNSAddress in SubjectLocality.

DFHSAML-TIMES container

DFHSAML-TIMES is a container of DATATYPE(CHAR). It contains a collection of time values.

Containers generated by CICS

CICS generates containers to store data such as variable arrays and long strings. These containers are created during pipeline processing and are used as input to, or output from, the application program. These containers are prefixed with DFH.

The naming convention for these containers is to use the CICS module that created them, combined with a numeric suffix to make the container name unique in the request. These container names occur during pipeline processing:

DFHPIAXIS-*nnnnnn*
: Containers that are used to store strings and variable arrays that are passed to the application in Axis2 pipelines. This container can also include binary data.

DFHPICC-*nnnnnnnn*
: Containers that are used to store strings and variable arrays that are passed to the application. This container can also include binary data.

DFHPIII-*nnnnnnnn*
: Outbound attachment containers created when the pipeline is enabled with the MTOM message handler and is running in direct mode. These containers are created when binary data is provided in a field rather than in a container by the application program.

DFHPIMM-*nnnnnnnn*
: Inbound attachment containers created during the processing of MIME messages. These containers are generated by CICS when the MTOM message handler is enabled in the pipeline. When direct mode processing is enabled, these containers can be passed through to the application directly.

DFHPIXO-*nnnnnnnn*
Outbound attachment containers created when the pipeline is enabled with the MTOM message handler and is running in compatibility mode.

The numbered container names start from 1 for each web service request; for example, DFHPICC-00000001. However, if an application program uses the **INVOKE SERVICE** command to initiate more than one web service request in the same channel, the containers that were returned to the application for one response might still exist when a further request is made. In this situation, CICS checks to see if the container already exists and increments the number of the generated container to avoid a naming conflict.

User containers

These containers contain information that 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.

Runtime processing for web services

To send a request to a web service provider or to receive a request from a web service requester, your application (or wrapper program) must interact correctly with the web services support in CICS. You can also control the processing that takes place in the pipeline to determine how the inbound and outbound requests are handled.

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.

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 SERVICE** command to invoke a web service. The request and response are mapped to a data structure in container DFHWS-DATA. This method of invoking a service is not supported for JSON.

Procedure

1. Create a channel and populate it with containers. At the minimum, container DFHWS-DATA must be present. DFHWS-DATA 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 the following command:

   ```
   EXEC CICS INVOKE SERVICE(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.

   For more information, see "Local optimization for web services" on page 321.
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 a 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 must decide if the application program should roll back any changes. If a SOAP fault 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.

Local optimization for web services

You can use the provider application name in the web service binding file associated with the WEBSERVICE resource to enable local optimization of the web service request.

Using the INVOKE SERVICE command, you can specify the URIMAP(urimap) or URI(uri) where the uri is the URI of the web service to be invoked. If a URIMAP is specified, CICS uses the client mode URIMAP indicated to resolves the URI. If these options are omitted, CICS uses the URI specified in the web service description (WSDL) from which the WEBSERVICE was generated.

If the WEBSERVICE indicated is deployed in a requester mode PIPELINE, CICS invokes the remote web service. This is the most typical scenario.

If the WEBSERVICE indicated is deployed in a provider mode PIPELINE, CICS invokes the service locally. If you use this optimization, the EXEC CICS INVOKE SERVICE command is optimized to an EXEC CICS LINK command. This results in significant performance benefits, but introduces the following limitations:

- The PIPELINE is not driven and therefore no handler programs are used.
- The PIPELINE control containers are not present on the Channel. Some containers are present, including the DFHWS-DATA, DFHWS-OPERATION and DFHWS-URI containers. Any containers that normally contain XML are not present; this includes the DFH-REQUEST, DFHWS-BODY and DFHWS-XMLNS containers.
- Both the provider and requester applications must share the same copybooks and be implemented in the same programming language.
- Both the provider and requester applications share a single unit of work.
- If the web service is not expected to return a response, the EXEC CICS INVOKE SERVICE command does not return control to the application until after the target PROGRAM has ended.

If you want to use locally optimized web services but require data to be processed through a PIPELINE, use the `cics` URI format described here: "Options for controlling requester pipeline processing" on page 325. This mechanism is less efficient than using the fully optimized approach, but it avoids the processing cost of going out to the network.

Runtime limitations for code generated by the web services assistant

At runtime, 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 WebSphere 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 application data is encoded in EBCDIC or UTF-16. 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. See Support for UTF-16 in application data for more information on UTF-16 support.

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 SERVICE** command, then any containers it passes to CICS could potentially be reused and the data within them replaced. If you want to keep the data in these containers, create a new channel and copy the containers to it before you run the program. 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 SERVICE** command, rather than using the default channel that it was originally attached to. If your application program is using the **INVOKE SERVICE** 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 runtime, 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 runtime.

- 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 trailing white space characters are removed when generating SOAP messages. At runtime, inbound SOAP messages are parsed according to the XML Schema specification and all leading, trailing, and embedded white space is removed.

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 runtime, 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 SERVICE** 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 runtime, then it is replaced with the equivalent entity. The XML entities that are supported are:

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.

Attention: CICS generates XML and JSON data from structured application data. If that application data contains bit patterns that look like preformatted XML, JSON, HTML, JPEG images, or any other meaningful content type, CICS is unaware of this semantic meaning, and processes such data as ordinary text or binary data. CICS does not attempt to recognize patterns in the data, or to process encoded data differently. For example, if the data contains pre-formatted XML (such as CDATA encoded text), that data is processed in the same way as any other data. Consider the following application data: `"An example: <here>"`. This example application-supplied data contains what looks like an XML tag, but it will be processed as raw text, resulting in the following XML representation: `"An example: < here > "`. If an application needs to generate XML itself, consider using `xsd:any` constructs in your XML schemas, or using XML-ONLY=TRUE in the Assistants.

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.

Customizing pipeline processing

In addition to providing your own message handlers, you can also use a set of global user exit points (GLUEs) to customize the processing that occurs for inbound and outbound web services in the pipeline.

Before you begin

You must understand the best practices for writing global user exit programs before customizing the pipeline. If you are customizing a service provider pipeline, you must be using the supplied DFHPITP or Axis2 application handler in your pipeline.

About this task

You can use the pipeline domain exits to access containers on a web services provider pipeline, a web services requester pipeline, or a web services requester pipeline that contains a security handler. The pipeline global user exits are described in detail in the *CICS Customization Guide*.

Procedure

1. Select which global user exit points to use:
   - Use XWSPRRWI, XWSPRROI, XWSPRROO, or XWSPRRWO GLUEs to access containers in a web services provider pipeline.
   - Use XWSRQRWO, XWSRQROO, XWSROROI, or XWSRQRWI GLUEs to access containers in a web services requester pipeline.
   - Use XWSSRRWO, XWSSRROO, XWSSRROI, or XWSSRRWI GLUEs to access containers in a secured web services requester pipeline.
2. Use the DFH$PIEX sample exit program to write your own global user exit program. DFH$PIEX is in the SDFHSAMP library. You are recommended to make the program threadsafe.
3. Enable the global user exit program.
4. Test your global user exit program to ensure it works correctly.

Related information:

Pipeline domain exits

The pipeline sample exit program

Writing global user exit programs

Defining, enabling, and disabling an exit program

Options for controlling requester pipeline processing

In service requester pipelines, message handlers can determine where the web service request is sent by changing the URI. CICS provides support for different URI formats so that you have much more flexibility in the way that the pipeline processes web service requests.

When the service requester pipeline reaches the end of its processing, you have the following options:

Linking to a program

If you change the URI to the format `cics://PROGRAM/`*program*, where *program* is the name of the target application program, CICS passes the current channel and its containers or COMMAREA to the program using an **EXEC CICS LINK** command.

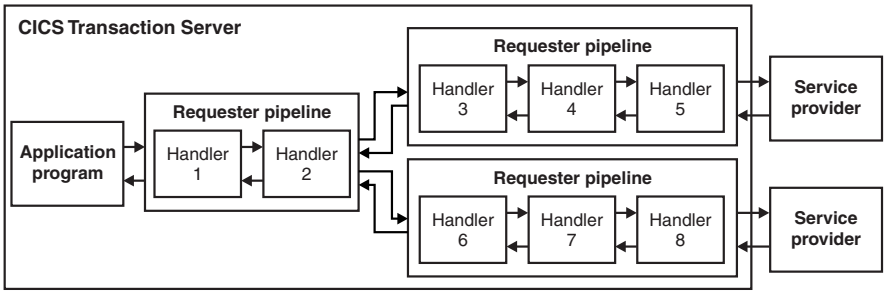
This processing is similar to the local optimization that occurs when the service requester and service provider applications are in the same CICS region. However, using this URI format provides the benefit of running the

request through the pipeline and any custom message handlers first. The target application program must be able to handle the contents of the containers or COMMAREA.

Starting another requester mode pipeline

If you change the URI to the format `cics://PIPELINE/pipeline?targetServiceUri=targetServiceUri`, where *pipeline* is the name of a PIPELINE resource and *targetServiceUri* is the URI that you want to put in the DFHWS-URI container, CICS passes the current channel and its containers to the specified requester pipeline. Use this URI when you want to link two or more requester pipelines together before sending the request to the service provider. The number of requester pipelines that you can chain together is not limited.

In the following example, one generic requester pipeline supports one application. Message handlers 1 or 2 can change the URI for each request depending on the application data in the containers, sending the request to one of two requester pipelines that contain different message handlers.



Although the example shows only one service requester application, many applications could use the same generic requester pipeline and have their requests sent to different requester pipelines before the request is finally sent to the appropriate web service provider.

Sending the request straight to the provider mode pipeline

If you change the URI to the format `cics://SERVICE/service?targetServiceUri=targetServiceUri`, where *service* is the name of the target service and *targetServiceUri* is the path to the service, CICS resolves the request by matching the path to a URIMAP and passes the request to the correct provider pipeline. Use this option when you want to take advantage of processing the request through both the requester and provider pipelines without using the network.

This URI might also be useful where the requester and provider applications are written in different languages, or use different mapping levels, and expect different binary data.

Related tasks:

"Controlling requester pipeline processing using a URI" on page 327
In service requester pipelines, a message handler can determine where to send the web service request by changing the URI. By changing the URI format, you can choose to perform certain optimizations, such as starting another requester pipeline or starting a service provider pipeline without sending the request over the network.

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the

service.

Controlling requester pipeline processing using a URI

In service requester pipelines, a message handler can determine where to send the web service request by changing the URI. By changing the URI format, you can choose to perform certain optimizations, such as starting another requester pipeline or starting a service provider pipeline without sending the request over the network.

Before you begin

Decide which options you want to implement in your requester pipeline. See "Options for controlling requester pipeline processing" on page 325 for details.

About this task

The web service requester application can populate the DFHWS-URI container using the **EXEC CICS INVOKE SERVICE** command or, if no value is supplied by the application, CICS populates the container using the value in the web service binding file. To modify the URI, you must write a message handler that changes the contents of this container.

Procedure

1. Write a message handler to modify the DFHWS-URI container according to one of the following URI formats:
   - To link to an application program, use the URI `cics://PROGRAM/`*`program`*, where *program* is the target application program. No data transformation takes place, so you must ensure that the application program can process the contents of the containers on the current channel. The application program can pass either the current channel and containers or a COMMAREA.
   - To start a provider pipeline without going through the network, use the URI `cics://SERVICE/`*`service`*`?targetServiceUri=`*`targetServiceUri`*, where *service* is the name of the service and *targetServiceUri* is the path of the service. The transport handler uses the path of the service to locate the URIMAP resource that resolves the request and passes it to the correct provider pipeline. CICS does not use the name of the service in its processing.

     An error occurs if no URIMAP resource is installed for the service. The URIMAP resource definition must also specify USAGE(PIPELINE). The transport handler puts the value of the **targetServiceUri** parameter in the DFHWS-URI container and starts the provider pipeline.
   - To start another requester pipeline, use the URI `cics://PIPELINE/`*`pipeline`*`?targetServiceUri=`*`targetServiceUri`*, where *pipeline* is the name of the PIPELINE resource that you want to start and *targetServiceUri* is the value that you want to pass to the next pipeline in the DFHWS-URI container.

   Each type of URI has additional parameters that you can add as a query string. For more information about the format of these URIs and the rules for coding them, see the "DFHWS-URI container" on page 304.
2. Use an XML editor to add the message handler to the pipeline configuration file:

```
<service>
  <service_handler_list>
    <handler>
```

```
      <program>MYPROG</program>
    </handler>
  </service_handler_list>
</service>
```

3. Disable, discard, and reinstall the PIPELINE resource for the requester pipeline to include your new message handler program in the pipeline.
4. Install the message handler program in the CICS region.

Results

The next service request to run through the requester pipeline is processed by your new message handler.

What to do next

Test out the changes to your requester pipeline to make sure that the service requests are going to the correct location and that your message handler program is behaving as designed.

Related concepts:

"Options for controlling requester pipeline processing" on page 325
In service requester pipelines, message handlers can determine where the web service request is sent by changing the URI. CICS provides support for different URI formats so that you have much more flexibility in the way that the pipeline processes web service requests.

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

JSON web service restrictions

Use this reference material to understand capabilities that are not supported by JSON web services.

The following capabilities are not supported:

- Requester mode pipelines with JSON are not supported.
- Runtime validation of JSON data against schema is not supported. The value of the VALIDATION attribute of a WEBSERVICE resource that is used with a JSON payload is ignored.
- Use of namespaces in JSON data (Badgerfish or Mapped conventions) is not supported.
- JSON payloads sent to CICS must be encoded in UTF-8. No other encoding is supported. Similarly, JSON sent by CICS is always encoded in UTF-8.
- WebSphere MQ transports with JSON pipelines are not supported.
- Vendor transformer programs are not supported for use with the JSON transformer.
- Reuse of WSBind files that are created for SOAP web services applications in a JSON pipeline is not supported. WSBind files for use with JSON service provider applications must be generated by the JSON assistant.
- If a JSON payload is missing some of its required content when CICS transforms it, the equivalent fields within the data structures are not initialized when passed to the application program.

- CICS cannot transform integer values greater than the maximum value for a signed long (2^63 - 1) unless they are enclosed within quotes.
- Use of simple data types or arrays is not supported at the root of a JSON Schema. The JSON Schema is required to describe a JSON Object, though the JSON Object can be composed of simple data types and arrays.
- If an array is declared in a JSON schema with a maxItems value of 1, CICS serializes the array as a simple string or integer when generating JSON at runtime.

Important: The only supported characters for JSON property names are: `A-Z a-z _ :` for the first character and `A-Z a-z 0-9 _ : . -` for all subsequent characters.

The Axis2 web services support today has a number of options for development and deployment of applications and customizations. The following options are not supported:

- User-supplied application handlers - you must use the CICS supplied application handler class `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler`.
- User-written Axis2 Java applications.
- The SOAPFAULT and WS-Addressing APIs cannot be used with the JSON pipeline.

Container restrictions

Note that some pipeline containers are not populated when processing a JSON request. For more information, see Containers used in the pipeline in Configuring.

Differences in RESTful web services

On INQUIRE PIPELINE:

- SOAPLEVEL returns NOTSOAP
- The MTOMNOXOPST, MTOMST, SENDMTOMST, SOAPRNUM, SOAPVNUM, XOPDIRECTST, and XOPSUPPORTST attributes are not used.

On INQUIRE WEBSERVICE:

- The ARCHIVEFILE, BINDING, VALIDATIONST, XOPDIRECTST, and XOPSUPPORTST attributes are not used.
- WSDLFILE returns the name of the JSON schema file that is associated with the WEBSERVICE.

On the WEBSERVICE resource:

- The ARCHIVEFILE and VALIDATION parameters are not used and their values are ignored.
- WSDLFILE is the name of the JSON schema file that is associated with the WEBSERVICE.

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 OASIS.

Note: CICS supports the November 2004 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.

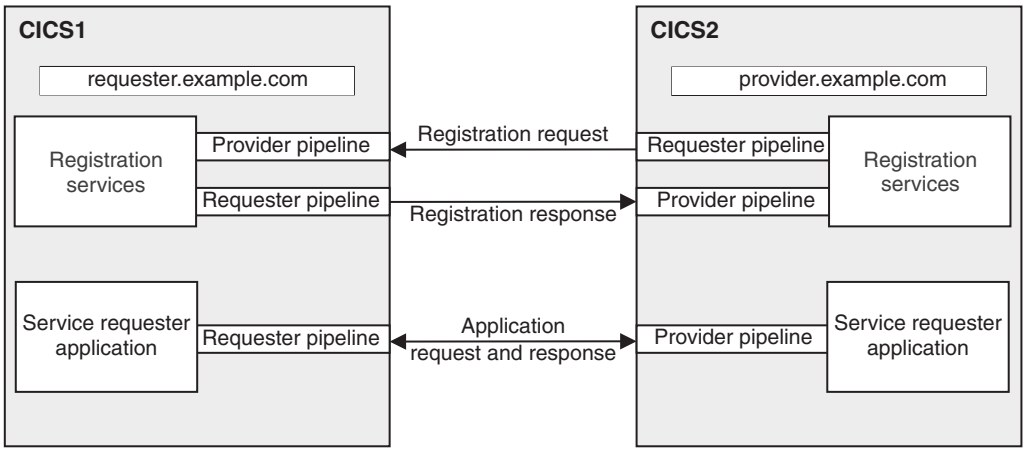


Figure 29. Registration services

Figure 29 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 30 on page 331.

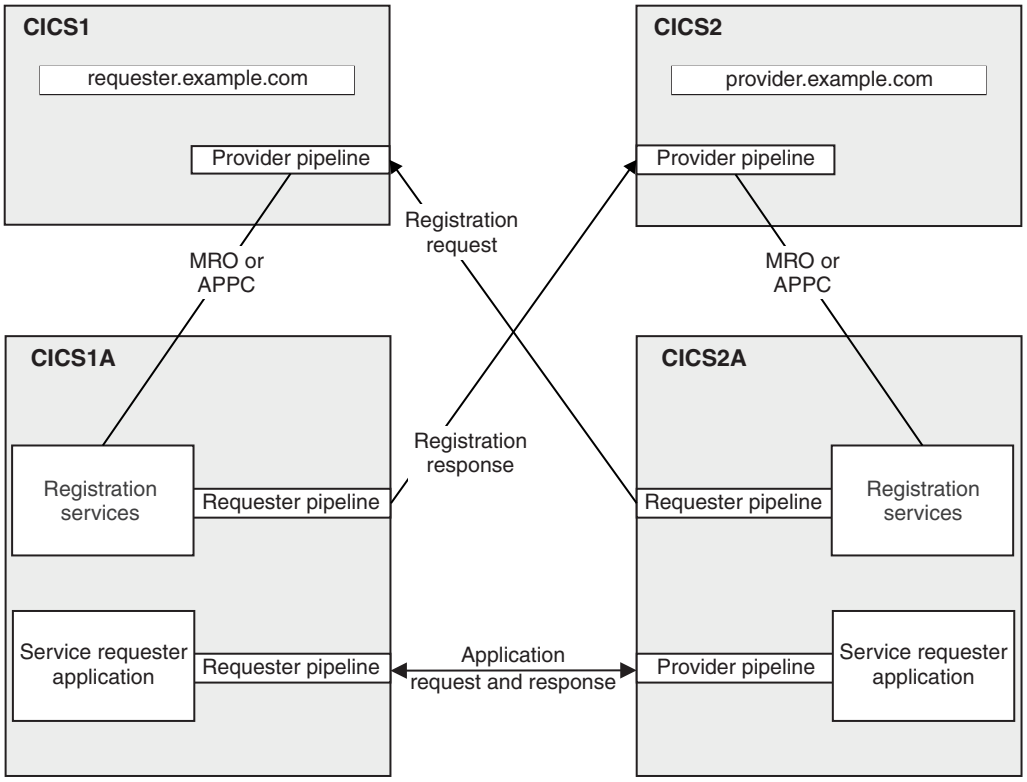


Figure 30. Registration services in a CICSPlex

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 30 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 begin

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/cicsts53/`

pipeline/configs directory, but the default file path might have been changed during CICS installation.

About this task

CICS support for web service transactions uses a CICS-supplied registration service. This registration service consists of a service provider and a service requester. You must install resources for both the service provider and the service requester; even if your applications are all service providers or all service requesters.

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 DFHWSAT 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:

Procedure

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. For more information about these groups, see Defining the WS-AT data set.

   If you want to share the DFHPIDIR file across CICS regions, the regions must be logically connected over MRO. You must install one data set per group of regions that are acting as a logical server.

   Tip: You are recommended not to share data sets between regions that are not logically connected.
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 registration service's provider PIPELINE resource. The PIPELINE is named DFHWSATP, and specifies the pipeline configuration file /usr/lpp/cicsts/cicsts53/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 registration service's requester PIPELINE resource. The PIPELINE is named DFHWSATR, and specifies the pipeline configuration file `/usr/lpp/cicsts/cicsts53/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.

Results

CICS is now configured so that your service provider and requester applications can participate in distributed transactions using WS-AtomicTransaction and WS-Coordination protocols.

What to do next

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>` element and a `<service_parameter_list>` element.

Before you begin

If you want your service provider application to 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.

About this task

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/cicsts53/samples/pipelines/wsatprovider.xml` directory (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX).

Procedure

1. In the definition of your terminal handler, code a `<headerprogram>` element in the `<cics_soap_1.1_handler>`, `<cics_soap_1.2_handler>`, `<cics_soap_1.1_handler_java>`, or `<cics_soap_1.2_handler_java>` 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/wscoor</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>
   ```

   address is the IP address of the CICS region where the registration service provider pipeline is located.

   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>
   http://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>` element and a `<service_parameter_list>` element.

Before you begin

If you want your service requester application to 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.

About this task

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/cicsts53/samples/pipelines/wsatrequester.xml directory (where /usr/lpp/cicsts/cicsts53 is the default install directory for CICS files on z/OS UNIX).

Procedure

1. Code a <headerprogram> element in the <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java>, or <cics_soap_1.2_handler_java> element. Code the <program_name>, <namespace>, <localname>, and <mandatory> elements exactly as shown in the following example:

   ```
   <cics_soap_1.1_handler>
     <headerprogram>
       <program_name>DFHWSATH</program_name>
       <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoor</namespace>
       <localname>CoordinationContext</localname>
       <mandatory>true</mandatory>
     </headerprogram>
   </cics_soap_1.1_handler>
   ```

   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>
   ```

   address is the IP address of the CICS region where the registration service provider pipeline is located.

   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 of 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.

About this task

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:

Procedure

- 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 <cicswsa: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.

Example

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">
        <cicswsa:Address xmlns:cicswsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
           http://clientIPaddress:clientPort/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
        </cicswsa:Address>
        <cicswsa:ReferenceProperties
           xmlns:cicswsa="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>
          </cicswsa: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 coordinator's 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.

About this task

If you need to access this information, you can either:

Procedure

- Look inside the contents of the `<cicswsa: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 `<cicswsa: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.

Example

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"
```

Support for MTOM/XOP optimization of binary data

In standard SOAP messages, binary objects are base64-encoded and included in the message body, which increases their size by 33%. For very large binary objects, this size increase can significantly impact transmission time. Implementing MTOM/XOP provides a solution to this problem.

The SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications, often referred to as MTOM/XOP, define a method for optimizing the transmission of large base64Binary data objects within SOAP messages.

- The MTOM specification conceptually defines a method for optimizing SOAP messages by separating out binary data, that would otherwise be base64 encoded, and sending it in separate binary attachments using a MIME Multipart/Related message. This type of MIME message is called an *MTOM message*. Sending the data in binary format significantly reduces its size, thus optimizing the transmission of the SOAP message.
- The XOP specification defines an implementation for optimizing XML messages using binary attachments in a packaging format that includes but is not limited to MIME messages.

CICS implements support for these specifications in both requester and provider pipelines when the transport protocol is WebSphere MQ, HTTP, or HTTPS. As an alternative to including the base64Binary data directly in the SOAP message, CICS applications that are deployed as web service providers or requesters can use this support to send and receive MTOM messages with binary attachments.

You can configure this support by using additional options in the pipeline configuration file.

MTOM/XOP and SOAP

When MTOM/XOP is used to optimize a SOAP message, it is serialized into a MIME Multipart/Related message using XOP processing. The base64Binary data is extracted from the SOAP message and packaged as separate binary attachments within the MIME message, in a similar manner to e-mail attachments.

The size of the base64Binary data is significantly reduced because the attachments are encoded in binary format. The XML in the SOAP message is then converted to XOP format by replacing the base64Binary data with a special `<xop:Include>` element that references the relevant MIME attachment using a URI.

The modified SOAP message is called the *XOP document*, and forms the root document within the message. The XOP document and binary attachments together form the *XOP package*. When applied to the SOAP MTOM specification, the XOP package is a MIME message in MTOM format.

The root document is identified by referencing its Content-ID in the overall content-type header of the MIME message. Here is an example of a content-type header:

```
Content-Type: Multipart/Related; boundary=MIME_boundary;
 type="application/soap+xml"; start="<claim@insurance.com>"
```

The **start** parameter contains the Content-ID of the XOP document. If this parameter is not included in the content-type header, the first part in the MIME message is assumed to be the XOP document.

The order of the attachments in the MIME message is unimportant. In some messages for example, the binary attachments could appear before the XOP document. An application that handles MIME messages must not rely on the attachments appearing in a specific order. For detailed information, read the MTOM/XOP specifications.

The following example demonstrates how a simple SOAP message that contains a JPEG image is optimized using XOP processing. The SOAP message is as follows:

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xmime="http://www.w3.org/2003/12/xop/mime">
 <soap:Body>
 <submitClaim>
  <accountNumber>5XJ45-3B2</accountNumber>
  <eventType>accident</eventType>
  <image xmime:contentType="image/jpeg" xsi:type="base64binary">4f3e..(encoded image)</image>
 </submitClaim>
 </soap:Body>
</soap:Envelope>
```

An MTOM/XOP version of this SOAP message is as follows:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
 type="application/soap+xml"; start="<claim@insurance.com>" 1

--MIME_boundary
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim@insurance.com> 2

<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xop='http://www.w3.org/2004/08/xop/include'
 xmlns:xop-mime='http://www.w3.org/2005/05/xmlmime'>
 <soap:Body>
 <submitClaim>
  <accountNumber>5XJ45-3B2</accountNumber>
  <eventType>accident</eventType>
  <image xop-mime:content-type='image/jpeg'><xop:Include href="cid:image@insurance.com"/></image> 3
 </submitClaim>
 </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <image@insurance.com> 4

...binary JPG image...

--MIME_boundary--
```

1. The **start** parameter indicates which part of the MIME message is the root XOP document.
2. The Content-ID value identifies a part of the MIME message. In this case it is the root XOP document.
3. The <xop:Include> element references the JPEG binary attachment.
4. The Content-ID identifies the JPEG in the binary attachment.

MTOM messages and binary attachments in CICS

CICS supports and controls the handling of MTOM messages in both web service provider and requester pipelines using an MTOM handler program and XOP processing.

You configure and enable the MTOM support using the pipeline configuration file. If MTOM support is enabled for a pipeline, CICS unpacks inbound MTOM

messages automatically and packages outbound messages. If MTOM support is not enabled for a pipeline and CICS receives an MTOM message, Java-based pipelines accept the inbound MTOM message, however other SOAP pipelines reject the inbound MTOM message with a SOAP fault.

Configuration options for Java-based pipelines

You can configure a provider pipeline to perform the following tasks:

- Accept MTOM messages, but never send MTOM response messages.
- Accept MTOM messages and always send MTOM response messages.
- Process XOP documents and binary attachments in Axis2 mode.

You can configure a requester pipeline to perform the following tasks:

- Never send an MTOM message, but accept MTOM response messages.
- Always send MTOM messages and accept MTOM response messages.
- Process XOP documents and binary attachments in Axis2 mode.

Configuration options for pipelines that do not support Java

You can configure a provider pipeline to perform the following tasks:

- Accept MTOM messages, but never send MTOM response messages.
- Accept MTOM messages and send the same type of response message.
- Accept MTOM messages, but only send MTOM messages when there are binary attachments present.
- Accept MTOM messages and always send MTOM response messages.
- Process XOP documents and binary attachments in direct or compatibility mode.

You can configure a requester pipeline to perform the following tasks:

- Never send an MTOM message, but accept MTOM response messages.
- Only send MTOM messages when there are binary attachments, and accept MTOM response messages.
- Always send MTOM messages and accept MTOM response messages.
- Process XOP documents and binary attachments in direct or compatibility mode.

Modes of support

There are three modes of support provided in the pipeline to handle XOP documents and any associated binary attachments.

Axis2 mode
: Axis2 mode is used when the terminal handler of your web services pipeline is either the <cics_soap_1.1_handler_java> or the <cics_soap_1.2_handler_java> message handler.

Direct mode
: In direct mode, the binary attachments associated with an inbound or outbound MTOM message are passed in containers through the pipeline and handled directly by the application, without the need to perform any data conversion.

Compatibility mode
: Compatibility mode is used when the pipeline processing requires the message to be in standard XML format, with any binary data stored as

base64Binary fields within the message. For inbound messages, the XOP document and binary attachments are reconstituted into a standard XML message, either at the beginning of the pipeline when Web Services Security is enabled, or at the end of the pipeline when web service validation is enabled. For outbound messages, a standard XML message is created and passed along the pipeline. This XML message is converted to XOP format by the MTOM handler just before CICS sends it.

Compatibility mode is much less efficient than direct mode because binary data gets converted to base64 format and back again. However, it does allow your web services to interoperate with other MTOM/XOP web service requesters and providers without needing to change your applications.

Inbound MTOM message processing for pipelines that do not support Java

When the MTOM handler is enabled in pipelines that do not support Java, it checks the headers of the inbound message in the DFHREQUEST or DFHRESPONSE container to determine the format of the message during the transport handling processing.

When a MIME Multipart/Related message is received, the MTOM handler unpackages the message as follows:

1. It puts the headers and binary data from each binary attachment into separate containers.
2. It puts the list of containers in the DFHWS-XOP-IN container.
3. It puts the XOP document, which formed the root of the message, back in the DFHREQUEST or DFHRESPONSE container, replacing the original message.

If there are no binary attachments, the XOP document is handled as a normal XML message and no XOP processing is required. If there are any binary attachments, XOP processing is enabled for the message.

If XOP processing is enabled, the MTOM handler checks the pipeline properties to determine if the current message should be processed in direct or compatibility mode, and puts this information in the DFHWS-MTOM-IN container.

In provider mode, the MTOM handler also creates the DFHWS-MTOM-OUT container to determine how the outbound response message should be processed.

Direct mode

When you are using CICS web services support, that is, when a service provider pipeline uses the DFHPITP application handler or a service requester pipeline is invoked using **INVOKE WEBSERVICE**, the pipeline can process the XOP document and binary attachments in direct mode.

In this mode, the XOP document and associated containers are passed by the MTOM handler to the next message handler in the pipeline for processing. The CICS web services support interprets the `<xop:Include>` elements. If the base64Binary field is represented as a container in the application data structure, then the attachment container name is stored in the structure. If the field is represented as a variable or fixed length string, the contents of the container are copied to the relevant application data structure field. The data structure is then passed to the application program.

Compatibility mode

If your pipeline is configured to use a custom application handler, or Web Services Security is also enabled, the message is processed in compatibility mode. In this mode, the XOP document and binary attachments are immediately reconstituted into a SOAP message using XOP processing, so that the content can be successfully processed in the pipeline. The XOP processing performs the following tasks:

1. Scans the XOP document for `<xop:Include>` elements, replacing each occurrence with the binary data from the referenced attachment in base64-encoded format.
2. Discards the DFHWS-XOP-IN container and all of the attachment containers.

The reconstituted SOAP message is then passed to the next handler in the pipeline to be processed as normal.

If web service validation is enabled, the pipeline switches to compatibility mode when the message reaches the application handler. The message is reconstituted into a SOAP message, validated, and passed to the application.

Outbound MTOM message processing for pipelines that do not support Java

When a pipeline that does not support Java is configured to send outbound MTOM messages, the web service and pipeline properties are checked to determine how the message should be processed and sent.

These properties are stored in two containers, DFHWS-MTOM-OUT and DFHWS-XOP-OUT. In a requester mode pipeline, these containers are created by CICS when the application issues the **EXEC CICS INVOKE WEBSERVICE** command. In a provider mode pipeline, the DFHWS-MTOM-OUT container is already initialized with the options that were determined when the inbound message was received.

If the outbound message can be processed in direct mode, the optimization of the message takes place immediately. If the outbound message has to be processed in compatibility mode, the optimization takes place at the very end of the pipeline processing.

If you have not deployed your web service provider or requester application using the CICS web services assistant, or if you have web service validation enabled or Web Services Security enabled in your pipeline, the outbound message is processed in compatibility mode.

Direct mode

In direct mode, the following processing takes place:

1. An XOP document is constructed from the application's data structure in container DFHWS-DATA. Any binary fields that are equal to or larger in size than 1500 bytes are identified, and the binary data and MIME headers describing the binary attachment are put in separate containers. If the binary data is already in a container, that container is used directly as the attachment. A `<xop:Include>` element is then inserted in the XML in place of the usual base64-encoded binary data using a generated Content-ID. For example:

   ```
   <xop:Include href="cid:generated-content-ID-value"
    xmlns:xop="http://www.w3.org/2004/08/xop/include">
   ```

2. All of the containers are added to the attachment list in the DFHWS-XOP-OUT container.

3. When the SOAP handler has processed DFHWS-DATA, the XOP document and SOAP envelope are stored in the DFHREQUEST or DFHRESPONSE container and processed through the pipeline.
4. When the last message handler has finished, the MTOM handler packages the XOP document and binary attachments into a MIME Multipart/Related message and sends it to the web service requester or provider. The DFHWS-XOP-OUT container and any associated containers are then discarded.

Compatibility mode

If the pipeline is not capable of handling the XOP document directly, then the following processing takes place:

1. The SOAP body is constructed in DFHWS-DATA from the application data structure and processed in the pipeline as normal.
2. When the final handler has finished processing the message, the MTOM handler checks the options in the DFHWS-MTOM-OUT container to determine whether MTOM should be used, optionally taking into account whether any binary attachments are present. If the MTOM handler determines that MTOM is not required, no XOP processing takes place and a SOAP message is sent by CICS as normal.
3. If the MTOM handler determines that the outbound message should be sent in MTOM format, the XOP processing scans the message for eligible fields to split the data out into binary attachments. For a field to be eligible, it must have the MIME **contentType** attribute specified on the element and the associated binary value must consist of valid base64Binary data in canonical form. The size of the data must be greater than 1500 bytes. The XOP processing creates the binary attachments and attachment list, and then replaces the fields with `<xop:Include>` elements.
4. The MTOM handler packages the XOP document and binary attachments as a MIME Multipart/Related message and CICS sends it to the web service requester or provider.

Restrictions when using MTOM/XOP

To support MTOM/XOP you can either specify the `<mtom>` element in your pipeline configuration file or enable the MTOM handler in your pipeline. However, there are restrictions associated with each method.

Restrictions for Java-based pipelines

Specifying the `<mtom>` element in the a pipeline configuration file enables MTOM/XOP support for your Java-based pipeline. However, there are restrictions with this MTOM/XOP implementation.

DFHPITP application handler
The Axis2 mode of MTOM/XOP support cannot be used with pipelines that specify DFHPITP as the application handler.

WS-Security
The Axis2 mode of MTOM/XOP support cannot be used with pipelines that use WS-Security configurations that require XML signatures.

Using the INQUIRE PIPELINE command
If an **INQUIRE PIPELINE** command is issued against a Java-based pipeline using the Axis2 mode of MTOM/XOP support, the **Mtomst**, **Sendmtomst**, **Mtomnoxopst**, **Xopsupportst**, and **Xopdirectst** attributes report as `Nomtom`. For more information, see INQUIRE PIPELINE.

Restrictions for other SOAP pipelines

Enabling the MTOM handler in the pipeline means that you can support web service implementations that use the MTOM/XOP optimization. The compatibility mode option means that you can interoperate with these web services without needing to change your web service applications. However, there are certain situations where you cannot use MTOM/XOP or its use is restricted.

Using the CICS web services assistant

The direct mode optimization for MTOM/XOP is only available if you are using DFHWS2LS at a mapping level of at least 1.2, and the WSDL document contains at least one field of type xsd:base64Binary. Web services that are enabled using DFHLS2WS are not eligible for XOP optimization.

Web services generated using DFHLS2WS with CHAR-VARYING=BINARY specified may be eligible for the MTOM/XOP optimizations. Other web services generated using DFHLS2WS do not contain binary data and are not eligible for the MTOM/XOP optimizations, but will work normally in a PIPELINE that supports MTOM/XOP.

Provider pipelines

CICS provides a default application handler called DFHPITP that can be configured in a provider pipeline. This application handler is capable of handling XOP documents and creating the necessary containers to support the pipeline processing in both direct and compatibility mode. If you are using your own application handler in a provider pipeline, and want to enable MTOM/XOP, you should configure the pipeline to run in compatibility mode.

Requester pipelines

If your applications use the **INVOKE WEBSERVICE** command, CICS handles the optimization of the SOAP message for you in direct and compatibility mode. If you are using the program DFHPIRT to start the pipeline, you can only send and receive MIME Multipart/Related messages in compatibility mode.

Web Services Security

If you enable the MTOM handler in the pipeline configuration file to run in direct mode, and you also enable the Web Services Security message handler, the pipeline only supports the handling of MTOM messages in compatibility mode.

Handling binary data

When you have large binary data to include in your web service, for example a graphic file such as a JPEG, you can use MTOM/XOP to optimize the size of the message that is sent to the service provider or requester. The minimum size of binary data that can be optimized using MTOM/XOP is 1500 bytes. If the binary data in a field is less than 1500 bytes, CICS does not optimize the field.

As stated in the XOP specification, there should be no white space in the base64Binary data. Any application programs that produce base64Binary data must use the canonical form. If the base64Binary data in an outbound message does contain white space, CICS does not convert the data to a binary attachment. When base64Binary data is generated by CICS, the fields are provided in canonical form and therefore contain no white space.

The **contentType** attribute must be present on base64Binary fields for XOP processing to occur in compatibility mode on outbound messages. The **contentType** attribute must not be present on hexBinary fields.

Web service validation

If you turn on web service validation the following pipeline processing takes place:

- If an inbound XOP document has been passed through the pipeline in direct mode, CICS automatically switches to compatibility mode and converts it back to standard XML when CICS web service support is about to validate the document.
- An outbound SOAP message is generated as standard XML and is processed in compatibility mode.

The extra pipeline processing is required because the validation processing cannot handle the contents of XOP documents.

Configuring CICS to support MTOM/XOP

To support MTOM messages in CICS, you must specify the correct MTOM/XOP support for your type of pipeline in your pipeline configuration files.

Configuring MTOM/XOP support for Java-based pipelines

To configure MTOM/XOP support for Java-based pipelines, you must add the `<mtom>` element to your pipeline configuration files.

Before you begin

Before performing this task, you must identify or create the pipeline configuration files to which you will add configuration information for MTOM/XOP.

About this task

If the `<mtom>` element is defined in your pipeline configuration file, MTOM support is enabled for all inbound and outbound messages. However, if this element is not specified in the pipeline configuration file, then MTOM support is enabled for only inbound messages.

Procedure

Add a `<mtom>` element to your pipeline configuration file. This element should be defined after the optional `<addressing>` element and before the optional `<headerprogram>` element.

Example

For a provider or requester mode pipeline, you could specify:

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <addressing></addressing>
  <mtom></mtom>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

Configuring MTOM/XOP for other SOAP pipelines

To configure MTOM/XOP support for pipelines that do not use the <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> handlers, you must add the MTOM handler to your pipeline configuration files.

Before you begin

Before performing this task, you must identify or create the pipeline configuration files to which you will add configuration information for MTOM/XOP.

Procedure

1. Add a <cics_mtom_handler> element to your pipeline configuration file. This element should be first in the <provider_pipeline> element, and the last element before the <service_parameter_list> in the <requester_pipeline> element. Code the following elements:

   ```
   <cics_mtom_handler>
     <dfhmtom_configuration version="1">
     </dfhmtom_configuration>
   </cics_mtom_handler>
   ```

   The <dfhmtom_configuration> element is a container for the other elements in the configuration. If you want to accept the default settings for MTOM/XOP processing, you can specify an empty element as follows:

   ```
   <cics_mtom_handler/>
   ```

2. Optional: Code an <mtom_options> element. In both a service provider and service requester pipeline, this element specifies whether the outbound message should be packaged as an MTOM message.
   a. Code the **send_mtom** attribute to define if the outbound message should be sent as an MTOM message. For details of this attribute, see "The <mtom_options> element" on page 259.
   b. Code the **send_when_no_xop** attribute to define if the outbound message should be sent as an MTOM message when there are no binary attachments present. For details of this attribute, see "The <mtom_options> element" on page 259.
3. Optional: Code a <xop_options> element with an **apphandler_supports_xop** attribute. This specifies if the application handler is capable of handling XOP documents directly. If you do not include this attribute, the default depends on whether the <apphandler> element specifies DFHPITP or another program. For details of this attribute, see "The <xop_options> element" on page 260.
4. Optional: Code a <mime_options> element with a **content_id_domain** attribute. This specifies the domain name that should be used when generating MIME content-ID values, that are used to identify binary attachments. For details of this attribute, see "The <mime_options> element" on page 261.

Example

The following example shows a completed <cics_mtom_handler> element in which all the optional elements are present:

```
<provider_pipeline>
  <cics_mtom_handler>
    <dfhmtom_configuration version="1">
      <mtom_options send_mtom="same" send_when_no_xop="no" />
      <xop_options apphandler_supports_xop="yes" />
      <mime_options content_id_domain="example.org" />
```

```
    </dfhmtom_configuration>
  </cics_mtom_handler>
....
</provider_pipeline>
```

Support for Web Services Addressing

CICS supports services that use the Worldwide Web Consortium (W3C) Web Services Addressing (WS-Addressing) specifications. This family of specifications provides transport-independent mechanisms to address web services and facilitate end-to-end addressing.

CICS ensures that your existing web service applications can accept requests from web services that use WS-Addressing. You can also create new web services that use endpoint references and message addressing properties in SOAP messages.

WS-Addressing adds addressing information, in the form of Message Addressing Properties (MAPs), to SOAP message headers. MAPs include messaging information, such as a unique message ID and endpoint references that detail where the message came from, where the message is going to, and where reply or fault messages are to be sent. An endpoint reference (EPR) is a specific type of MAP, which includes the destination address of the message, optional reference parameters for use by the application, and optional metadata.

Features of the WS-Addressing support

CICS includes the following features to support WS-Addressing:

- Your web service requester and provider applications can interact with other services that are using WS-Addressing without requiring you to redeploy them. A new message handler, the addressing message handler DFHWSADH, in the pipeline routes messages that contain WS-Addressing information to the specified web service.
- You can write an application that uses the WS-Addressing API commands to create an endpoint reference and to create, update, delete, and query an addressing context.
- You can route response messages to endpoints other than the requester endpoint; for example, you can route fault messages to a dedicated fault handler.
- You can pass reference parameters to applications as part of the MAPs in the SOAP header.

Support for WS-Addressing specifications and interoperability

By default, CICS supports the recommendation specifications:

- W3C WS-Addressing 1.0 - Core
- W3C WS-Addressing 1.0 - SOAP Binding
- W3C WS-Addressing 1.0 - Metadata

These specifications are identified by the `http://www.w3.org/2005/08/addressing` namespace. Unless otherwise stated, WS-Addressing semantics that are described in this documentation refer to the recommendation specifications.

For interoperability, CICS also supports the submission specification:

- W3C WS-Addressing- Submission

This specification is identified by the `http://schemas.xmlsoap.org/ws/2004/08/addressing` namespace. Use the submission specification only if you must interoperate with a client or web service provider that implements the submission specification.

Web Services Addressing overview

Web Services Addressing (WS-Addressing) provides a standard framework for specifying the endpoints of a SOAP message. This framework is transport-neutral and improves the interoperability of web services that use different transport mechanisms. The WS-Addressing specification introduces message addressing properties and endpoint references.

Web Services Addressing (WS-Addressing) is a Worldwide Web Consortium (W3C) specification that improves interoperability between web services by defining a standard way to address web services and provide addressing information in SOAP messages. SOAP messages can be sent over a variety of transport mechanisms, including HTTP and WebSphere MQ, each of which stores destination information for the message in a different way.

Existing CICS web services that are deployed in a pipeline configured to use WS-Addressing can use the default WS-Addressing settings without requiring any changes. To take full advantage of the WS-Addressing capabilities, use the WS-Addressing API commands. The WS-Addressing implementation supports one SOAP fault for each WSDL operation.

Message addressing properties

Message addressing properties (MAPs) are a set of well defined WS-Addressing properties that can be represented as elements in SOAP headers. MAPs provide a standard way of conveying information, such as the endpoint to which message replies must be directed, or information about the relationship that the message has with other messages. The MAPs that are defined by the WS-Addressing specification are summarized in the following table.

Table 12. Message addressing properties defined by the WS-Addressing specification

Abstract WS-Addressing MAP name	SOAP WS-Addressing MAP name	MAP content type	Multiplicity	Description
[action]	<wsa:Action>	xs:anyURI	1..1	An absolute URI that uniquely identifies the semantics of the message. This value is required.
[destination]	<wsa:To>	xs:anyURI in the SOAP message EndpointReference in the addressing context	0..1	The absolute URI that specifies the address of the intended receiver of the message. If this value is not specified, it defaults to the anonymous URI that is defined in the specification: `http://www.w3.org/2005/08/addressing/anonymous`. In the addressing context, the <wsa:To> MAP is represented as an EPR. When the <wsa:To> is sent as part of a SOAP message it is split into its address and its reference parameters, as defined by the schema.

Table 12. Message addressing properties defined by the WS-Addressing specification (continued)

Abstract WS-Addressing MAP name	SOAP WS-Addressing MAP name	MAP content type	Multiplicity	Description
[reference parameters] *	[reference parameters]*	xs:any	0..unbounded	Parameters that correspond to <wsa:ReferenceParameters> properties of the endpoint reference to which the message is addressed. This value is optional.
[source endpoint]	<wsa:From>	EndpointReference	0..1	A reference to the endpoint from which the message originated. This value is optional.
[reply endpoint]	<wsa:ReplyTo>	EndpointReference	0..1	An endpoint reference for the intended receiver of replies to this message. This value is optional. If this value is not specified, it defaults to `http://www.w3.org/2005/08/addressing/anonymous`.
[fault endpoint]	<wsa:FaultTo>	EndpointReference	0..1	An endpoint reference for the intended receiver of faults relating to this message. This value is optional and defaults to the value of the <wsa:ReplyTo> MAP.
[relationship] *	<wsa:RelatesTo>	xs:anyURI plus optional attribute of type xs:anyURI	0..unbounded	A pair of values that indicate how this message relates to another message. The contents of this element conveys the <wsa:MessageID> of the related message. An optional attribute conveys the relationship type. This value is optional. If this value is not specified, it defaults to `http://www.w3.org/2005/08/addressing/reply`.
[message id]	<wsa:MessageID>	xs:anyURI		An absolute URI that uniquely identifies the message. This value is optional; if not supplied, CICS generates a value for outbound requests and responses.

The following example of a SOAP message contains WS-Addressing MAPs:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://w3.org/2005/08/addressing"
            xmlns:example="http://example.ibm.com/namespace">
  <S:Header>
      ...
      <wsa:To>http://example.ibm.com/enquiry</wsa:To>
      <wsa:ReplyTo>
        <wsa:Address>http://example.ibm.com/enquiryReply</wsa:Address>
      </wsa:ReplyTo>
      <wsa:Action>...</wsa:Action>
      <example:AccountCode wsa:IsReferenceParameter='true'>123456789</example:AccountCode>
      <example:DiscountId wsa:IsReferenceParameter='true'>ABCDEFG</example:DiscountId>
      ...
  </S:Header>
  <S:Body>
  ...
  </S:Body>
</S:Envelope>
```

Endpoint references

An endpoint reference is a specific type of MAP, which provides a standard mechanism to encapsulate information about specific endpoints. Endpoint

references can be sent to other parties and used to target the web service endpoint that they represent. The following table summarizes the information model for endpoint references.

Table 13. Information model for endpoint references

Abstract property name	Property type	Multiplicity	Description
[address]	xs:anyURI	1..1	The absolute URI that specifies the address of the endpoint.
[reference parameters] *	xs:any	0..unbounded	Namespace qualified element information items that are required to interface with the endpoint.
[metadata]	xs:any	0..unbounded	Description of the behavior, policies, and capabilities of the endpoint.

The following XML fragment illustrates an endpoint reference. The `<wsa:EndpointReference>` element references the endpoint at the URI `http://example.ibm.com/enquiry` and contains metadata specifying the interface to which the endpoint reference refers and some application-specific reference parameters.

```
<wsa:EndpointReference
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:example="http://example.ibm.com/namespace">
  <wsa:Address>http://example.ibm.com/enquiry</wsa:Address>
  <wsa:Metadata
      xmlns:wsdli="http://www.w3.org/ns/wsdl-instance"
      wsdli:wsdlLocation="http://example.ibm.com/wsdl/wsdl-location.wsdl">
    <wsam:InterfaceName>example:reservationInterface</wsam:InterfaceName>
  </wsa:Metadata>
  <wsa:ReferenceParameters>
      <example:AccountCode>123456789</example:AccountCode>
      <example:DiscountId>ABCDEFG</example:DiscountId>
   <wsa:ReferenceParameters>
</wsa:EndpointReference>
```

WS-Addressing MAPs of type `wsa:EndpointReferenceType` are: <wsa:From>, <wsa:ReplyTo>, and <wsa:FaultTo>. However, the <wsa:To> MAP is defined in the WS-Addressing 1.0 standard as having a type of `xs:anyURI`. For simplicity CICS treats <wsa:To> MAPs in the addressing context as EPRs. When a <wsa:To> MAP is sent as part of a SOAP message, CICS splits it into its address and reference parameters, as required by the standard.

Default namespaces

The following prefix and corresponding namespaces are referred to throughout the WS-Addressing documentation:

Table 14. Prefix and corresponding namespace

Default prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
wsa	http://www.w3.org/2005/08/addressing (Recommendation schema) http://schemas.xmlsoap.org/ws/2004/08/addressing (Submission schema)

Table 14. Prefix and corresponding namespace (continued)

Default prefix	Namespace
wsam	http://www.w3.org/2007/05/addressing/metadata

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

Configuring a requester pipeline for Web Services Addressing

To configure a requester pipeline to support Web Services Addressing (WS-Addressing), you must add an addressing handler to your pipeline configuration file.

Before you begin

You must identify or create the pipeline configuration file to add the configuration information for WS-Addressing. You must also decide which of the WS-Addressing specifications to use. Use the *W3C WS-Addressing 1.0 Core* specification where possible.

About this task

You can add support for WS-Addressing in one of two ways:

- If the SOAP pipeline uses Java, the SOAP processing is handled by Axis2 and you can use the support provided by this technology to handle requests that use WS-Addressing. All of the header handling is handled by Axis2 and it is important that you do not add the DFHWSADH header processing program to the pipeline. You can use your own header processing programs. For better performance, write Axis2 handlers in Java if you want to process SOAP headers.
- If the SOAP pipeline does not use Java, you must add the CICS-supplied header processing program DFHWSADH to handle the requests.

Procedure

- If the SOAP pipeline uses a `<cics_soap_1.1_handler_java>` or `<cics_soap_1.2_handler_java>` element, add an `<addressing>` element to the pipeline configuration file. Include one `<namespace>` element that contains the specification that you want to use on the request message, which can be different to the response message; for example, you can always send a request that complies with the W3C core specification, even if the response message uses the submission specification. Axis2 supports both WS-Addressing specifications on inbound messages.

  The following example shows how you might configure the requester pipeline:

```
<requester_pipeline>
   <service>
      <service_handler_list>
         <cics_soap_1.1_handler_java>
            <jvmserver>JVMSERV1</jvmserver>
            <addressing>
               <namespace>http://www.w3.org/2005/08/addressing</namespace>
            </addressing>
         </cics_soap_1.1_handler_java>
      </service_handler_list>
   </service>
</requester_pipeline>
```

The <jvmserver> element contains the name of the JVMSERVER resource that supports Axis2.

- If the SOAP pipeline does not use Java, add a CICS addressing header program in the <cics_soap_1.1_handler> or <cics_soap_1.2_handler> to the pipeline configuration file. The following example shows how you might configure the requester pipeline:

```
<requester_pipeline>
   <service>
      <service_handler_list>
         <cics_soap_1.1_handler>
            <headerprogram>
               <program_name>DFHWSADH</program_name>
               <namespace>http://www.w3.org/2005/08/addressing</namespace>
               <localname>*</localname>
               <mandatory>true</mandatory>
            </headerprogram>
         </cics_soap_1.1_handler>
      </service_handler_list>
   </service>
</requester_pipeline>
```

  Code the <program_name>, <localname>, and <mandatory> elements exactly as shown. Set <namespace> to http://www.w3.org/2005/08/addressing to use the *W3C WS-Addressing 1.0 Core* specification or http://schemas.xmlsoap.org/ws/2004/08/addressing to use the *W3C WS-Addressing Submission* specification.

  The order of header processing programs is not guaranteed. If you define other header processing programs, add them in a subsequent CICS SOAP handler element in your <service_handler_list> element. The DFHWSADH header handler must be in the first SOAP handler element.

Results

Your requester pipeline is now configured to support WS-Addressing.

What to do next

Create a PIPELINE resource that points to the configuration file. If you are using a Java-based SOAP pipeline, ensure that a JVMSERVER resource is enabled to handle the Axis2 processing.

Configuring a provider pipeline for Web Services Addressing

To configure a provider pipeline to support Web Services Addressing (WS-Addressing), you must add an addressing handler to your pipeline configuration file.

Before you begin

You must identify or create the pipeline configuration file to add the configuration information for WS-Addressing. You must also decide which of the WS-Addressing specifications to use. Use the *W3C WS-Addressing 1.0 Core* specification where possible.

About this task

You can add support for WS-Addressing in one of two ways:

- If the SOAP pipeline uses Java, the SOAP processing is handled by Axis2 and you can use the support provided by this technology to handle requests that use

WS-Addressing. All of the header handling is handled by Axis2 and it is important that you do not add the DFHWSADH header processing program to the pipeline. You can use your own header processing programs. For better performance, write Axis2 handlers in Java if you want to process SOAP headers.

- If the SOAP pipeline does not use Java, you must add the CICS-supplied header processing program DFHWSADH to handle the requests.

Procedure

- If the SOAP pipeline uses a <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> element, add an <addressing> element to the pipeline configuration file. You can optionally include one or more <namespace> elements. This element contains the specification that you want to use on the outbound message, which can be different to the inbound message; for example, you can always send an outbound response that complies with the W3C core specification, even if the inbound message uses the submission specification. If you exclude this element, Axis2 uses the same specification on the outbound message as the inbound message. Axis2 supports both WS-Addressing specifications on inbound messages.

  The following example shows how you might configure the provider pipeline:

```
<provider_pipeline>
   <terminal_handler>
      <cics_soap_1.1_handler_java>
          <jvmserver>JVMSERV1</jvmserver>
          <addressing>
             <namespace>http://www.w3.org/2005/08/addressing</namespace>
          </addressing>
      </cics_soap_1.1_handler_java>
    </terminal_handler>
</provider_pipeline>
```

  The <jvmserver> element contains the name of the JVMSERVER resource that supports Axis2.

- If the SOAP pipeline does not use Java, add the CICS addressing header program DFHWSADH to the SOAP handler in the pipeline configuration file. The following example shows how you might configure the provider pipeline:

```
<provider_pipeline>
   <terminal_handler>
      <cics_soap_1.1_handler>
         <headerprogram>
             <program_name>DFHWSADH</program_name>
             <namespace>http://www.w3.org/2005/08/addressing</namespace>
             <localname>*</localname>
             <mandatory>true</mandatory>
         </headerprogram>
      </cics_soap_1.1_handler>
   </terminal_handler>
</provider_pipeline>
```

  Code the <program_name>, <localname>, and <mandatory> elements exactly as shown. Set <namespace> to http://www.w3.org/2005/08/addressing to use the *W3C WS-Addressing 1.0 Core* specification or http://schemas.xmlsoap.org/ws/2004/08/addressing to use the *W3C WS-Addressing Submission* specification.

  The order of header processing programs is not guaranteed. If you define other header processing programs, add them in another CICS SOAP handler element in a <service_handler_list> element. The DFHWSADH header handler must be in the last SOAP handler element.

Results

Your provider pipeline is now configured to support WS-Addressing.

What to do next

Create a PIPELINE resource that points to the configuration file. If you are using a Java-based SOAP pipeline, ensure that a JVMSERVER resource is enabled to handle the Axis2 processing.

Creating a web service that uses WS-Addressing

To create a web service from a WSDL document that uses Web Services Addressing (WS-Addressing), use parameters on the web services assistant to handle the conversion from XML to language structures.

About this task

You can use the web services assistant job, DFHWS2LS, to control how an end point reference (EPR) is handled in the WSDL document and determine whether CICS constructs default input, output, and fault actions.

Procedure

1. Set the **MINIMUM-RUNTIME** parameter on the web services assistant, DFHWS2LS, to `3.0` or higher. A runtime level of at least 3.0 ensures that any generated web service binding fully supports WS-Addressing and can interoperate with other web services platforms.
2. Set the **MAPPING-LEVEL** parameter on the web services assistant, DFHWS2LS, to `3.0` or higher.
3. Set the **WSADDR-EPR-ANY** parameter to `TRUE` if you want to use `wsa:EndpointReferenceType` type elements in the request or response messages. End point references can be included in application data and you have the option of using the EPR in API commands such as **WSACONTEXT BUILD**. Setting the **WSADDR-EPR-ANY** parameter to `TRUE` indicates that CICS must not transform the EPR into a language structure at run time; instead, CICS treats the EPR data as an <xsd:any> element and stores it in a named container.

   This example WSDL fragment shows a <wsa:To> MAP being passed as an element of type `wsa:EndpointReferenceType`:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="exampleEPR" targetNamespace="http://example.ibm.com/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:s0="http://example.ibm.com/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  <types>
    <xs:schema targetNamespace="http://test.org/"
            xmlns:s="http://www.w3.org/2001/XMLSchema"
            xmlns:s0="http://example.ibm.com/"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
      ...
       <xs:element name="exampleResponse" type="s0:typeResponse"/>
       <xs:complexType name="typeResponse">
         <xs:sequence>
           <xs:element name="myEpr" type="wsa:EndpointReferenceType"/> 1
         </xs:sequence>
       </xs:complexType>
       ...
    </xs:schema>
```

```
  </types>
  ...
  <message name="msgResponse">
    <part element="s0:exampleResponse" name="response"/>
  </message>
  ...
</definitions>
```

When the element, `<xs:element name="myEpr" type="wsa:EndpointReferenceType"/>` **1**, is processed by DFHWS2LS with the **WSADDR-EPR-ANY** parameter set to TRUE, the myEpr element data is stored in a named container as an <xsd:any> element and a pointer to the container added to the generated language structure.

For example, the COBOL language structure generated by DFHWS2LS for the myEpr element is shown here:

```
09 myEpr.
   12 myEpr-xml-cont               PIC X(16).
   12 myEpr-xmlns-cont             PIC X(16).
```

The myEpr-xml-cont container stores the name of the container that contains the myEpr data. The myEpr-xmlns-cont is an optional container that is populated with any XML namespace declarations that are in scope.

4. Save and submit the DFHWS2LS job.

Results

CICS creates a web service binding to handle the data transformation and language structures that you can use to create the service requester or provider application.

What to do next

To enable the web service, perform a pipeline scan to create the required CICS resources.

Default end point references

Most WSDL documents contain the address at which the web service is hosted. In WS-Addressing, the WSDL document can also contain an end point reference (EPR) for the web service. This EPR can contain additional metadata to facilitate communication between the requester and provider applications.

If you use DFHWS2LS to process the WSDL, the EPR is saved in the web service binding and is used by CICS to send request and response messages. Any reference parameters, <wsa:ReferenceParameters>, that are defined in the EPR are included in the SOAP message. This EPR is known as the default EPR, because it can be overridden by the application. If the application does not supply an explicit EPR, the default EPR from the WSDL is used.

The following WSDL 1.1 fragment includes a default EPR: `<soap:address location="http://example.ibm.com:12345/exampleTest" />`. The <port> element includes a child element, <wsa:EndpointReference>, the address specified by the child element, **2**, must match the address specified by the parent element, **1**:

```
<service name="exampleService">
   <port name="examplePort" binding="s0:createBinding">
      <soap:address location="http://example.ibm.com:12345/exampleTest" /> 1

      <wsa:EndpointReference
             xmlns:example="http://example.ibm.com/namespace"
```

```
                        xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
                        wsdli:wsdlLocation="http://example.ibm.com/location "
                        title="http://example.ibm.com/example/example_wsdl"
                        class="http://example.ibm.com/example/example.wsdl">
                        <wsa:Address>http://example.ibm.com:12345/exampleTest</wsa:Address> 2
                        <wsa:Metadata>
                           <wsam:InterfaceName>example:Inventory</wsam:InterfaceName>
                        </wsa:Metadata>
                        <wsa:ReferenceParameters>
                          <example:AccountCode>123456789</example:AccountCode>
                          <example:DiscountId>ABCDEFG</example:DiscountId>
                        </wsa:ReferenceParameters>
                    </wsa:EndpointReference>
                </port>
            </service>
```

Explicit actions

WSDL documents can explicitly define the values of the <wsa:Action> properties. If the WSDL document does not contain explicitly defined <wsa:Action> properties, CICS builds default actions when the WSDL is processed by DFHWS2LS.

WSDL 1.1

The following WSDL 1.1 fragment represents a booking system that contains explicitly defined <wsa:Action> properties:

```
<definitions targetNamespace="http://example.ibm.com/namespace" ...>
   ...
   <portType name="bookingSystem">
      <operation name="makeBooking">
         <input message="tns:makeBooking"
            wsa:Action="http://example.ibm.com/namespace/makeBooking"
         </input>
         <output message="tns:bookingResponse"
             wsa:Action="http://example.ibm.com/namespace/bookingResponse"
         </output>
      </operation>
   </portType>
   ...
</definitions>
```

In this example, the input action of the `makeBooking` operation is explicitly defined as `http://example.ibm.com/namespace/makeBooking`, and the output action is explicitly defined as `http://example.ibm.com/namespace/bookingResponse`.

WSDL 2.0

The following WSDL 2.0 fragment represents a booking system that contains explicitly defined <wsa:Action> properties:

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <interface name="bookingInterface">
    <operation name="makeBooking" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input element="tns:makeBooking" messageLabel="In"
            wsa:Action="http://example.ibm.com/namespace/makeBooking"/>
      <output element="tns:makeBookingResponse" messageLabel="Out"
            wsa:Action="http://example.ibm.com/namespace/makeBookingResponse"/>
    </operation>
  </interface>
  ...
</description>
```

In this example, the input action of the makeBooking operation is explicitly defined as http://example.ibm.com/namespace/makeBooking, and the output action is defined as http://example.ibm.com/namespace/makeBookingResponse.

For more information, see the W3C WS-Addressing 1.0 Metadata specification.

Default actions for WSDL 1.1

If a WSDL 1.1 document does not contain explicitly specified <wsa:Action> properties, CICS builds default input, output, and fault actions when the WSDL is processed by DFHWS2LS.

Default input and output actions for WSDL 1.1

The following pattern is used by CICS in WSDL 1.1 documents that follow either the recommendation schema or the submission schema to construct a default input or output action:

```
[target namespace]/[port type name]/[input|output name]
```

Default fault actions for WSDL 1.1

If you are following the recommendation schema, the way that CICS builds the default fault action differs from the behavior described in the schema. The following pattern is used by CICS, in WSDL 1.1 documents that follow the recommendation schema, to construct a default fault message. Notice that the fault name is omitted.

```
[target namespace]/[port type name]/[operation name]/Fault/
```

If you are following the submission schema, the way that CICS builds the default fault action follows the behavior described in the schema. The following pattern is used by CICS, in WSDL 1.1 documents that follow the submission schema, to construct a default fault message:

```
[target namespace]/[port type name]/[operation name]/Fault/[fault name]
```

Example of the default actions generated by CICS for a WSDL 1.1 document

This example of a booking system illustrates how CICS constructs default actions from a WSDL 1.1 document:

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <portType name="bookingInterface">
    <operation name="makeBooking">
      <input element="tns:makeBooking" name="MakeBooking"/>
      <output element="tns:bookingResponse" name="BookingResponse"/>
         <fault message="tns:InvalidBooking" name="InvalidBooking"/>
    </operation>
  </interface>
  ...
</definitions>
```

The WSDL fragment has the following addressing properties:

Property name	Value
[targetNamespace]	http://example.ibm.com/namespace
[portType name]	bookingInterface
[operation name]	makeBooking

Property name	Value
[input name]	`MakeBooking`
[output name]	`BookingResponse`
[fault name]	`InvalidBooking`

The following actions are created from these values:

Action	Value
Input Action	`http://example.ibm.com/namespace/bookingInterface/MakeBooking` If the [input name] is not specified, the value of the [operation name] with "Request" appended is used instead. For example, in this case the Input Action is `http://example.ibm.com/namespace/bookingInterface/makeBookingRequest`.
Output Action	`http://example.ibm.com/namespace/bookingInterface/BookingResponse` If the [output name] is not specified, the value of the [operation name] with "Response" appended is used instead. For example, in this case the Output Action is `http://example.ibm.com/namespace/bookingInterface/makeBookingResponse`
Fault Action (Recommendation schema)	`http://example.ibm.com/namespace/bookingInterface/MakeBooking/Fault/` Notice that the [fault name] is omitted.
Fault Action (Submission schema)	`http://example.ibm.com/namespace/bookingInterface/MakeBooking/Fault/InvalidBooking`

For more information, see the W3C WS-Addressing 1.0 Metadata specification.

Default actions for WSDL 2.0

If a WSDL 2.0 document does not contain explicitly specified <wsa:Action> properties, CICS build default input, output, and fault actions when the WSDL is processed by DFHWS2LS.

Default input and output actions for WSDL 2.0

The following pattern is used by CICS, in WSDL 2.0 documents that follow the recommendation schema, to construct default actions for inputs and outputs:

```
[target namespace]/[interface name]/[operation name][direction token]
```

Default fault actions for WSDL 2.0

If you are following the recommendation schema, the way that CICS builds the default action for WS-Addressing faults differs from the behavior described in the schema. If you are following the submission schema, the way that CICS builds the default action for WS-Addressing faults follows the behavior described in the schema.

The following pattern is used by CICS, in WSDL 2.0 documents that follow the recommendation schema, to construct a default action for faults. Notice that the fault name is omitted.

```
[target namespace]/[interface name]/
```

The following pattern is used by CICS, in WSDL 2.0 documents that follow the submission schema, to construct a default action for faults:

```
[target namespace]/[interface name]/[fault name]
```

Example of the default actions generated by CICS for a WSDL 2.0 document

This example shows how CICS constructs default actions for a WSDL 2.0 document following the recommendation schema:

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <interface name="bookingInterface">
    <operation name="makeBooking" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input element="tns:makeBooking" messageLabel="In"/>
      <output element="tns:bookingResponse" messageLabel="Out"/>
    </operation>
  </interface>
  ...
</definitions>
```

The WSDL fragment has the following addressing properties:

Property Name	Value
[targetNamespace]	`http://example.ibm.com/namespace`
[interface name]	`bookingInterface`
[operation name]	`makeBooking`
[direction token]	Either `Request` or `Response`.

The following input and output actions are created from these values:

Action	Value
Input Action	http://example.ibm.com/namespace/bookingInterface/makeBookingRequest
Output Action	http://example.ibm.com/namespace/bookingInterface/makeBookingResponse

For more information, see the W3C WS-Addressing 1.0 Metadata specification.

Message exchanges

Web Services Addressing (WS-Addressing) supports these message exchanges: one-way, two-way request-response, synchronous request-response, and asynchronous request-response.

Web Services Addressing message exchanges involve message addressing properties (MAPs) and endpoint references (EPRs).

At run time CICS ensures that the SOAP header of the request message contains the relevant WS-Addressing message information, the requester application does not have to set the WS-Addressing headers and might not even be aware that it is using WS-Addressing.

One-way

This straightforward one-way message is defined as an input-only operation. The web Services Description Language (WSDL) for this operation takes the following form:

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
</operation>
```

If you are using WS-Addressing, CICS adds the <wsa:Action> MAPs and the <wsa:MessageID> MAP to the SOAP message header of the WS-Addressing request message at run time to ensure compliance with the WS-Addressing specification.

The <wsa:MessageID> MAP is a unique ID, if not specified CICS generates this ID automatically.

The <wsa:Action> MAPs are derived from the WSDL and stored in the WSBind file.

You can override the values of these MAPs using the CICS WS-Addressing API commands.

Two-way request-response

This two-way exchange involves a request message and a response message. The response part of the operation can be defined as an output message, a fault message, or both. The WSDL definition for a request-response operation takes the following form:

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
  <output message="tns:myOutputMessage"/>
  <fault="tns:myFaultMessage"/>
</operation>
```

Responses to, or faults generated from, requests that are directed at endpoints are targeted at the <wsa:ReplyTo> MAP or the <wsa:FaultTo> MAP depending on whether the reply type is normal or a fault.

Specify a <wsa:ReplyTo> or <wsa:FaultTo> MAP in the request message to indicate where the response must be sent.

If you are using the recommendation specifications and do not specify a value for the <wsa:ReplyTo> MAP, the <wsa:ReplyTo> MAP defaults to an endpoint reference that contains the anonymous URI (http://www.w3.org/2005/08/addressing/anonymous), which causes CICS to send the response back to the requester.

If you are using the recommendation specifications and do not specify a value for the <wsa:FaultTo> MAP, the <wsa:FaultTo> MAP defaults the value of the <wsa:ReplyTo> MAP.

If the requester builds MAPs that are incorrect and that cause validation failures, CICS sends the fault message back to the requester instead of to the address specified by the <wsa:FaultTo> MAP.

Synchronous request-response

By default, the response part of a two-way message is returned according to the underlying protocol in use. In the case of an HTTP request, the response is returned synchronously in the HTTP response.

Asynchronous request-response

An asynchronous response is targeted at another web service and does not arrive back at the original requester application. In the case of an HTTP request, the connection with the requesting client is closed with an HTTP 202 response. If the web service provider is running on a CICS system, the requester application will receive an empty response message. If the web service provider is running on a WebSphere MQ system, the requester application will not receive any response.

To change the destination of the response part of a two-way message, you must specify the appropriate addresses in the <wsa:ReplyTo> MAP, or the <wsa:ReplyTo> and <wsa:FaultTo>, MAPs.

For a full list of the MAPs that are mandatory in WSDL 1.1 and WSDL 2.0, see "Mandatory message addressing properties for WS-Addressing."

Related concepts:

"WSDL and message exchange patterns" on page 30
A WSDL 2.0 document contains a message exchange pattern (MEP) that defines the way that SOAP 1.2 messages are exchanged between the web service requester and web service provider.

Related reference:

"Mandatory message addressing properties for WS-Addressing"
The WS-Addressing 1.0 metadata specification states which message addressing properties (MAPs) must be included in WSDL 1.1 and WSDL 2.0 documents. The CICS implementation of WS-Addressing helps you to comply with the WS-Addressing specifications by automatically supplying values for these mandatory MAPs.

Mandatory message addressing properties for WS-Addressing

The WS-Addressing 1.0 metadata specification states which message addressing properties (MAPs) must be included in WSDL 1.1 and WSDL 2.0 documents. The CICS implementation of WS-Addressing helps you to comply with the WS-Addressing specifications by automatically supplying values for these mandatory MAPs.

You can specify your own values for MAPs in the WSDL that you supply, and you can update these values in the addressing context using the CICS WS-Addressing API commands. If you do not supply values for the mandatory MAPs, CICS will generate values for you.

The following table lists which MAPs are mandatory for the different supported message exchange patterns (MEPs) with WSDL 1.1 and WSDL 2.0:

Table 15. Mandatory message addressing properties for WS-Addressing.

WS-Addressing MAP name	Description	Mandatory in WSDL 1.1	Mandatory in WSDL 2.0
<wsa:To>	The address of the intended receiver of the message.	No	No

Table 15. Mandatory message addressing properties for WS-Addressing. (continued)

WS-Addressing MAP name	Description	Mandatory in WSDL 1.1	Mandatory in WSDL 2.0
<wsa:Action>	The WS-Addressing action: input, output, or fault.	Mandatory for the following MEPs: One-way Two-way (Request) Two-way (Response)	Mandatory for the following MEPs: In-only Robust In-only (In) Robust In-only (Fault) In-out (In) In-out (Out) In-optional-out (In) In-optional-out (Out)
<wsa:From>	The endpoint from which the message originated.	No	No
This	value	not	required
<wsa:ReplyTo>	The endpoint of the intended receiver for replies to the message.	No	No
<wsa:FaultTo>	The endpoint of the intended receiver for faults related to the message.	No	No
<wsa:MessageID>	A unique message identifier.	Mandatory for the following MEPs: Two-way (Request)	Mandatory for the following MEPs: Robust In-only (In) In-out (In) In-optional-out (In)
<wsa:RelatesTo>	A pair of values that indicate how this message relates to another message. This element includes the <wsa:MessageID> of the related message and an optional attribute conveys the relationship type.	Mandatory for the following MEPs: Two-way (Response)	Mandatory for the following MEPs: Robust In-only (Fault) In-out (Out) In-optional-out (Out)

For more information, see the *W3C WS-Addressing 1.0 Metadata* specification: http://www.w3.org/TR/ws-addr-metadata.

Notes:

- If a value is not set for the address element of the <wsa:ReplyTo> MAP, the address is set to the anonymous URI: `http://www.w3.org/2005/08/addressing/anonymous`. The anonymous URI indicates that responses are sent back to the requester.
- If a value is not specified for the address element of the <wsa:FaultTo> MAP, CICS sets this address to the same value as the address element of the <wsa:ReplyTo> MAP.

  Note that if the requester builds MAPs that are incorrect and which cause validation failures, CICS sends the fault message back to the requester instead of to the address specified by the <wsa:FaultTo> MAP.
- If the value of the <wsa:To> MAP is not specified, CICS set the address to the anonymous URI: `http://www.w3.org/2005/08/addressing/anonymous`. The

anonymous URI indicates that the request is to be sent to the address specified in the DFHWS-URI container; for more information, see "DFHWS-URI container" on page 304.

- You can define the <wsa:Action> MAPs explicitly in your WSDL document, or you can let CICS generate them automatically.
- CICS automatically sets a unique value for the <wsa:MessageID> MAP at run time for request messages that expect a response, and for response messages.
- The <wsa:RelatesTo> MAP is mandatory for response messages. The relationship type of the message is optional and defaults to http://www.w3.org/2005/08/addressing/reply.

Related concepts:

"Message exchanges" on page 359
Web Services Addressing (WS-Addressing) supports these message exchanges: one-way, two-way request-response, synchronous request-response, and asynchronous request-response.

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

Web Services Addressing security

Communications traveling on a public network using Web Services Addressing (WS-Addressing) must be adequately secured and a sufficient level of trust must be established between the communicating parties. You are recommended to use transport level security, such as SSL or HTTPS, to secure your communications.

Transport level security, such as SSL or HTTPS, is the most straightforward way to ensure that your WS-Addressing communications are secure. If transport level security is not available, you can secure your messages by signing the WS-Addressing message addressing properties and encrypting the endpoint references.

CICS cannot sign headers containing WS-Addressing message addressing properties or encrypt endpoint references. However, CICS can verify signatures on incoming messages and can decrypt headers that have been encrypted. If you want to use signing and encryption to secure your communications, you must use an external security gateway, such as the IBM WebSphere DataPower® XML Security Gateway. For more information, see IBM WebSphere DataPower XML Security Gateway.

Web Services Addressing example

This example provides a high-level overview of the process that takes place when a customer places an order with a company that uses Web Services Addressing to send messages.

An international company that sells electronic components uses Web Services Addressing in its business. The infrastructure of this company consists of an Ordering Client, a group of Distribution Services, a Fulfilment Service, and a Configuration Service.

Using WS-Addressing offers the company the following benefits:

- WS-Addressing provides a transport-independent mechanism for transferring messages, this encourages interoperability between web services running on

different platforms. In this example, the distribution services owned by the company are running on a variety of platforms; WS-Addressing makes interoperability between different platforms straightforward because the web service requesters and providers do not need to be aware of the platform on which the service that they are exchanging messages with is running.

- WS-Addressing can be used to change the destination of the reply message by updating the EPR in the <wsa:ReplyTo> MAP. In this example, the Fulfilment Service modifies the destination of the response message when it selects the Distribution Service to which the message is diverted.

The company has several distribution centers in a number of different countries; each of the distribution centers is represented in this example by a Distribution Service and is registered with the Configuration Service.

The Fulfilment Service selects which Distribution service is the most appropriate to process the order based on a variety of factors, which might include the availability of items requested and the distance of the Distribution Center from the customer.

Addressing information is passed to and from the Configuration Service. The Configuration Service stores the addresses of the available services in the form of Endpoint References. New services register with the Configuration Service by creating an EPR using the **WSAEPR CREATE** command and sending the EPR to the Configuration Service. The Configuration Service requires the EPR as a block of XML, so the **WSADDR-EPR-ANY** parameter on DFHWS2LS must be set to TRUE. The **WSADDR-EPR-ANY=TRUE** option is used to instruct CICS to treat the EPR as an <xsd:any> element; CICS must place it in a container instead of transforming it into a language structure at run time.

The way in which these services interact is shown in the following diagram. The diagram shows other services, which have been excluded from the task, that might be relevant in a business application:

- A Tracking Service, which can be updated by each of the other services with the status of the order.
- A Problem Resolution service to handle any fault messages that arise.
- An Ordering Client callback service to handle any reply messages directed at the Ordering Client.

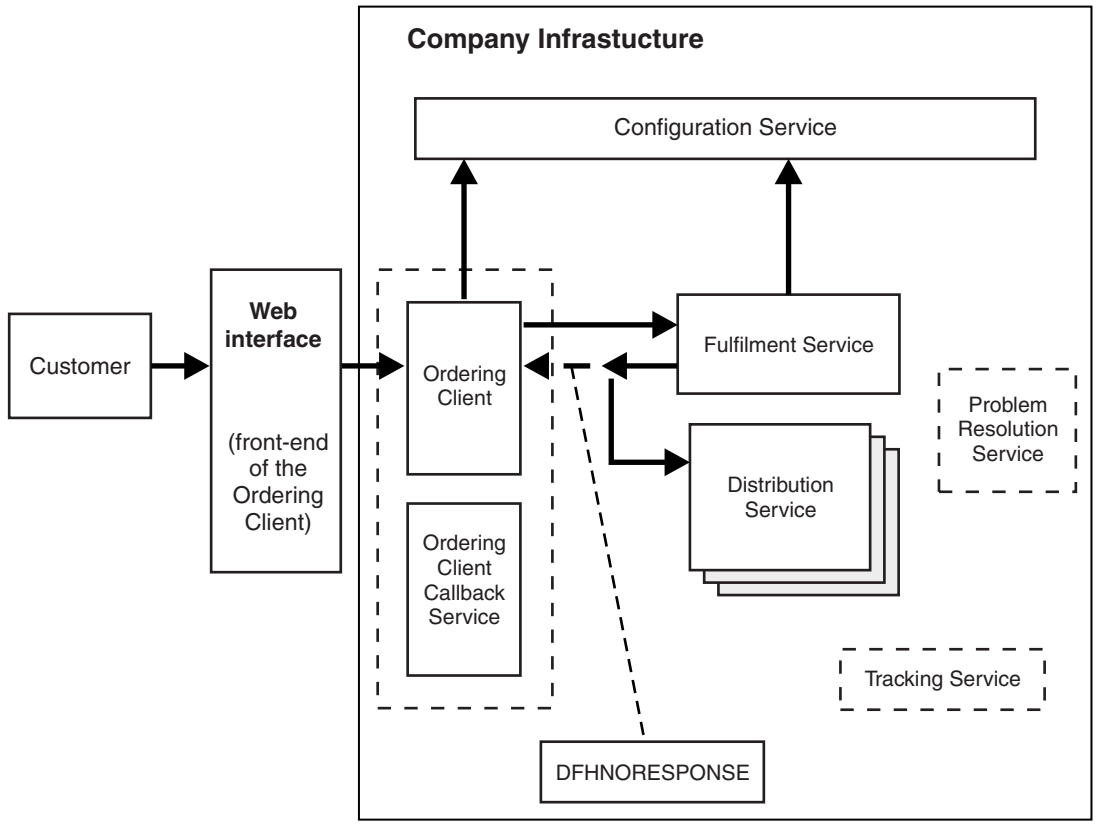


Figure 31. Company infrastructure

The following steps describe the process that takes place from the time a customer places an order to the point at which that order is processed.

1. A customer places an order with the company.
   a. The customer places the order on the company website, which is the front end for the Ordering Client.
   b. The Ordering Client takes the customer's contact details as part of the order.
   c. The Ordering Client returns a confirmation and a unique order reference to the customer through the web interface.
2. The Ordering Client sends the order request to the Fulfilment Service.
   a. If the Ordering Client does not already know the EPR for the Fulfilment Service, it requests it from the Configuration Service. The process involved when the Ordering Client requests the EPR of the Fulfilment Service from the Configuration service is detailed in the Example of <wsa:To> section.
   b. The Ordering Client issues the **INVOKE SERVICE** command for the Fulfilment Service. WS-Addressing routes the message to the address specified by the To EPR in the request addressing context.
3. The Fulfilment Service selects a Distribution Service to process the order and redirects the response message to that service.
   a. The Fulfilment Service uses a **WSACONTEXT GET** command to extract the order reference and other addressing properties from the addressing context.
   b. The Fulfilment Service selects the most appropriate Distribution Service from the Configuration Service.
   c. The <wsa:ReplyTo> EPR is added to the addressing context:

```
<wsa:EndpointReference
     xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsa:Address>http://www.example.ibm.com/DistributionService</wsa:Address>
  </wsa:EndpointReference>
```

The Fulfilment Service uses the **WSACONTEXT BUILD** command to add the ReplyTo EPR of the chosen Distribution Service to the request addressing context.

d. The Fulfilment Service uses the **WSACONTEXT BUILD** command repeatedly to add the order reference and other information to the request addressing context.

e. A DFHNORESPONSE container is added to the Ordering Client pipeline to indicate to the Ordering Client that it will not receive a response and the response message is redirected in the form of a request message to the Distribution Service.

4. The Distribution Service receives the redirected response message and processes the order.
   a. The Distribution Service uses a **WSACONTEXT GET** command to extract the order reference and addressing details from the request addressing context.
   b. The Distribution Service process the order.

Example of <wsa:To>

1. The Ordering Client requests the EPR of the service that it wants to send a message to from the Configuration Service. In this example, the Ordering Client requests the EPR of the Fulfilment Service.
2. The Configuration Service creates and sends a response message:
   a. The Configuration Service creates the requested <wsa:To> EPR for the Fulfilment Service using the **WSAEPR CREATE** API command: `EXEC CICS WSAEPR CREATE`.
   b. The Configuration Service writes the output from the **WSAEPR CREATE** command to a container: `EXEC CICS PUT CONTAINER(work-cont)`.
   c. The Configuration Service copies the container name into the `myEpr-xml-cont` element: `MOVE work-cont TO myEpr-xml-cont`.
   d. The Configuration Service sends a response message to the Ordering Client, this message contains the contents of the container named by the `myEpr-xml-cont` container. In this example, the contents of the `work-cont` container is sent to the Ordering Client inside the `<wsa:myEpr>` element:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   ...
   <env:Body>
      <wsa:myEpr>
         <wsa:EndpointReference>
            <wsa:Address>
          Fulfilment_Service_EPR_XML
            </wsa:Address>
         </wsa:EndpointReference>
      </wsa:myEpr>
   </env:Body>
   ...
</env:Envelope>
```

Figure 32 on page 367 shows the request-response message exchange between the Ordering Client and the Configuration Service. This message exchange involves two typical web services pipelines.

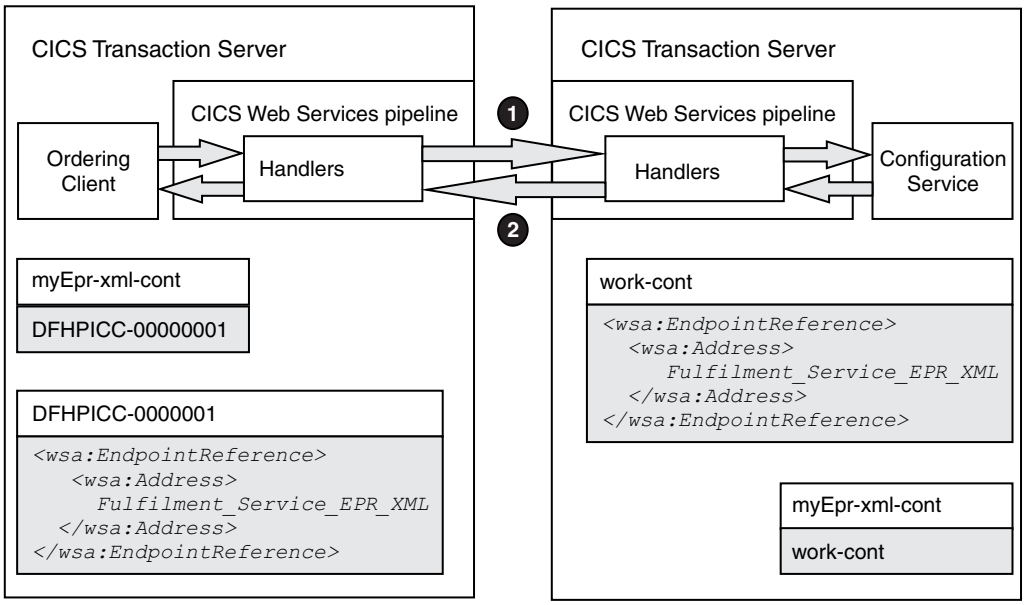


Figure 32. Request-response message exchange between the Ordering Client and the Configuration Service

3. The Ordering Client receives the response message, builds the <wsa:To> EPR, and sends a request to the Fulfilment Service:
   a. The Ordering Client extracts the <wsa:To> EPR data from the response message.
   b. CICS populates a unique container, in this example the `DFHPICC-00000001` container, with the <wsa:To> EPR data.
   c. CICS copies the name of the container, in this example `DFHPICC-00000001`, into the `myEpr-xml-cont` element.
   d. The Ordering Client reads the contents of the container specified by the `myEpr-xml-cont` element and provides it as input to the **WSACONTEXT BUILD** API command. The **WSACONTEXT BUILD** command uses this input to build the <wsa:To> EPR for the Fulfilment Service.
   e. The Ordering Client issues an **INVOKE SERVICE** command which initiates the pipeline processing.
   f. The CICS web services addressing handler, DFHWSADH, on the outbound pipeline converts the <wsa:To> EPR into an address and an optional set of reference parameters which it puts into the header of the SOAP request message that is being sent to the Fulfilment Service:

```
<env:Header>
  <wsa:To>http://example.ibm.com/Fulfilment_Service</wsa:To>
</env:Header>
```

Figure 33 on page 368 shows the request from the Ordering Client to the Fulfilment service. This request involves a web services pipeline that includes the CICS web services addressing handler, DFHWSADH.

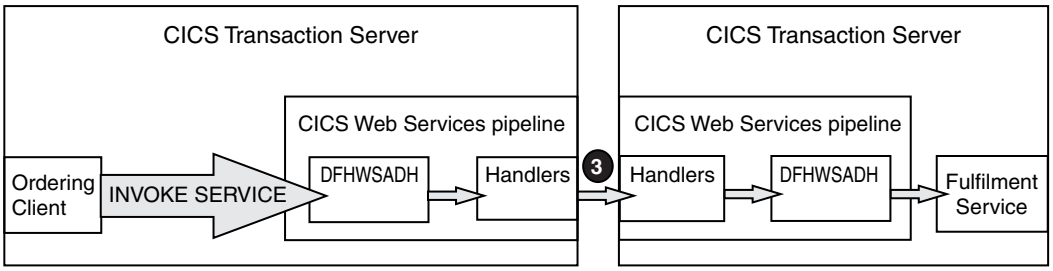


Figure 33. Request from the Ordering Client to the Fulfilment Service

Related reference:

"Web Services Addressing terminology"
Terms used to explain Web Services Addressing (WS-Addressing) support.

Web Services Addressing terminology

Terms used to explain Web Services Addressing (WS-Addressing) support.

addressing context
: An XML document that stores WS-Addressing message addressing properties (MAPs) before they are sent in SOAP request messages and after they are received from SOAP request and response messages.

endpoint reference (EPR)
: An XML structure containing addressing information that is used to route a message to a web service. This addressing information includes the destination address of the message, optional reference parameters for use by the application, and optional metadata.

message addressing property (MAP)
: An XML element that conveys addressing information for a specific web service message, such as a unique message ID, the destination of the message, and the endpoint references of the message.

Support for SAML

CICS supports the use of Security Assertion Markup Language (SAML) for describing and exchanging security information between online business partners.

CICS supports the SAMLCore1.1 and SAML Core2.0 standards. It does not support the protocols that are described in those standards.

You can configure provider and requester pipelines to use SAML tokens, but you must first deploy the CICS Security Token Service (STS). For more information about configuring your CICS installation to support SAML, see Configuring CICS for SAML.

Configuring CICS web services for Kerberos

To configure a provider pipeline to implement Kerberos authentication, you must add a security handler to your pipeline configuration file.

Before you begin

You must configure an external security manager, such as RACF, to enable support for Kerberos. For more information, see Configuring RACF for Kerberos.

You must identify or create the pipeline configuration file to add the configuration information for Kerberos.

Procedure

1. Add a <wsse_handler> element to your pipeline. The handler must be included in the <service_handler_list> element. 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.
2. Code an <authentication> element.
   a. Code the **trust** attribute to specify `trust="basic"`.
   b. Code the **mode** attribute to specify `mode="basic-kerberos"`.
   c. Optional: Code an empty <suppress/> element. If you do not specify this element, the work runs under the user ID associated with the token.

Results

The CICS provider pipeline is configured for Kerberos authentication. Inbound web service requests received by the pipeline must include a valid Kerberos token. If they do not, the request is rejected with the appropriate SOAP fault. Depending on pipeline configuration options, the target application runs under the user ID associated with the token.

Example

The following example shows how you might configure the provider pipeline:

```
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
    <service>
        <service_handler_list>
            <wsse_handler>
                <dfhwsse_configuration version="1">
                   <authentication trust="basic" mode="basic-kerberos"/>
                </dfhwsse_configuration>
            </wsse_handler>
        </service_handler_list>
        <terminal_handler>
            <cics_soap_1.1_handler/>
        </terminal_handler>
    </service>
    <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Chapter 7. Creating a SOAP web service

You can expose existing CICS applications as SOAP web services and create new CICS applications to act as SOAP web service providers or requesters.

Before you begin

Before you begin to create a SOAP web service, perform these tasks:

1. Configure your CICS system to support web services; see "Configuring your CICS system for web services" on page 51.
2. Create the necessary infrastructure to support the deployment of your web services; see Chapter 6, "Creating the web services infrastructure," on page 51.
3. Decide whether you want to use the web services assistant; see "Planning to use SOAP web services" on page 45.

About this task

The CICS web services assistant is a supplied utility that helps you to create the necessary artifacts for a new SOAP web service provider or a service requester application, or to enable an existing application as a web service provider.

The CICS web services assistant can create a WSDL document from a simple language structure or a language structure from an existing WSDL document; it supports COBOL, C/C++, and PL/I. It also generates information that is used to enable automatic runtime conversion of the SOAP messages to containers and COMMAREAs, and vice versa. This information is used by the CICS web services support during pipeline processing.

Create your web service, as described in the following procedure, and validate that it works correctly:

Procedure

1. Create a SOAP web service in one of four ways:
   - Use the web services assistant to create the web service description or language structures and deploy them into CICS. Use the **PIPELINE SCAN** command to automatically create the required CICS resources.
   - Use Rational Developer for z Systems or the Java API to create the web service description or language structures and deploy them into CICS. Use the **PIPELINE SCAN** command to automatically create the required CICS resources.
   - Create or change an application program to handle the XML in the inbound and outbound messages, including the data conversion, and populate the correct containers in the pipeline. You must create the required CICS resources manually.
   - Deploy an Axis2 application as a web service.
2. Start the web service to test that it works as you intended. If you are using the web services assistant to deploy your web service, you can use the **SET WEBSERVICE** command to turn on validation. This validation checks that the data is converted correctly.

What to do next

These steps are explained in more detail in the following topics.

The CICS web services assistant

The CICS web services assistant is a set of batch utilities that 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.

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.

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.

For more information on the web services assistant's utility programs and data mappings, see the following topics.

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.

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 Starts the job.

EXEC Specifies the procedure name (DFHLS2WS).

INPUT.SYSUT1 DD
: Specifies the input. The input parameters are typically 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 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/ to produce a complete path name of /usr/lpp/*path*.

Typically, you do not 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 z/OS UNIX directory path used on other parameters. The default is the empty string.

Typically, you do not 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 z/OS UNIX 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 file system. The value of this parameter is appended to /usr/lpp/cicsts/ to produce a complete path name of /usr/lpp/cicsts/*path*.

Typically, you do not 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 at run time:

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 as follows:

/tmp/LS2WS.in

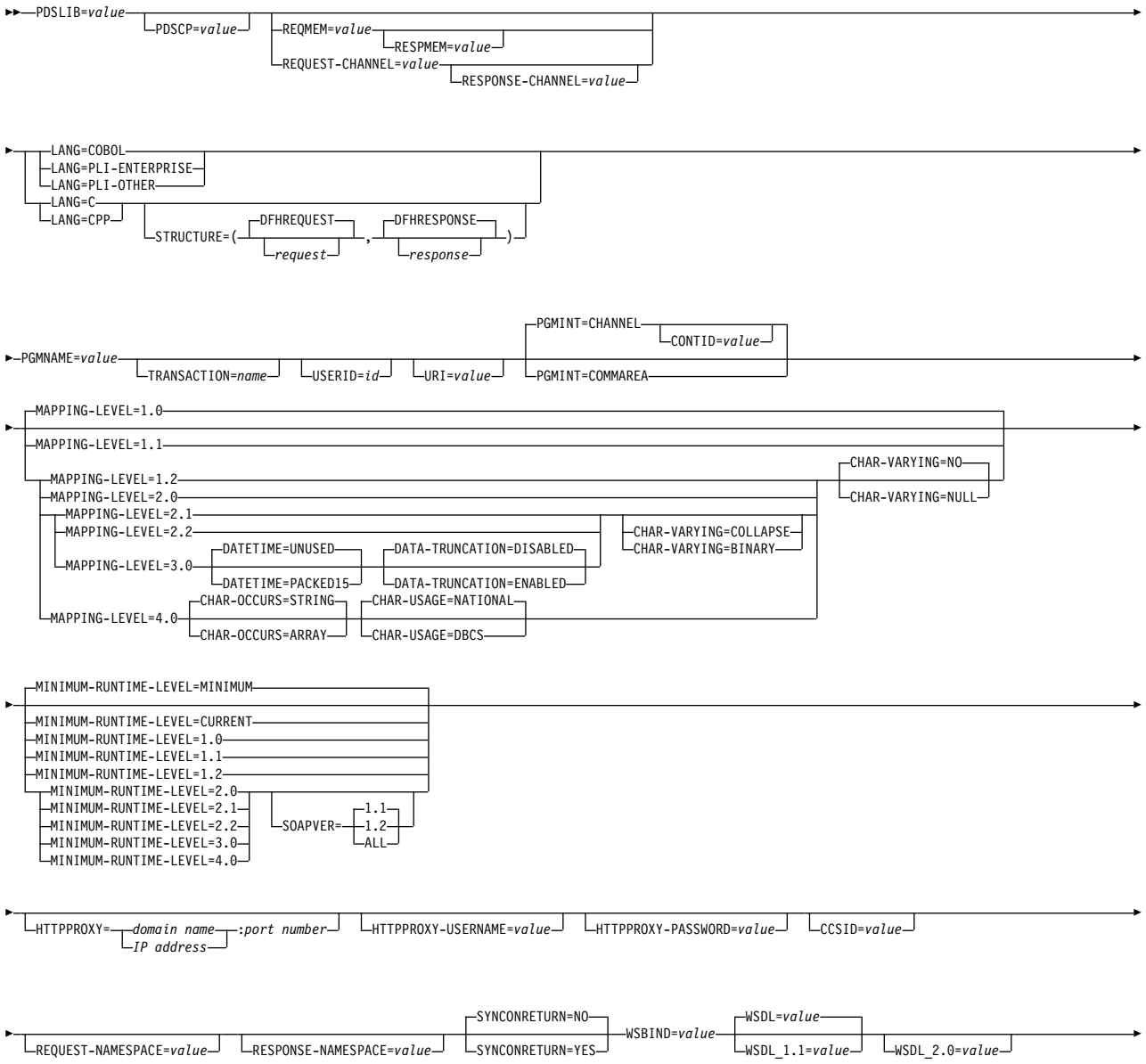
/tmp/LS2WS.out

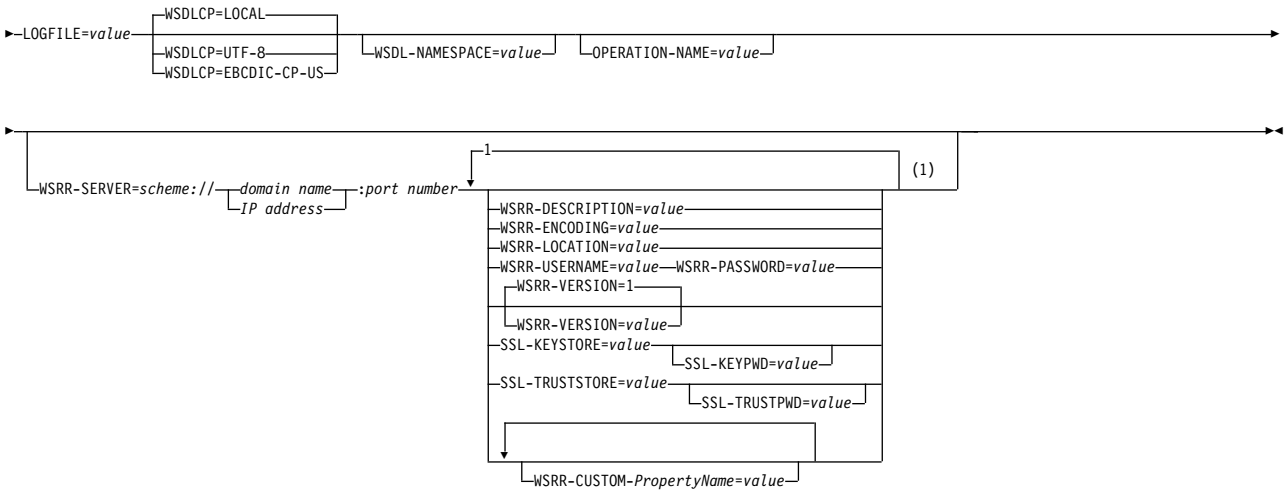
```
/tmp/LS2WS.err
```

Important: DFHLS2WS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHLS2WS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that 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





Notes:

1 Each of the WSRR parameters that can be specified when the **WSRR-SERVER** parameter is set can be specified only once. The exception to this rule is the **WSRR-CUSTOM** parameter, which you can specify a maximum of 255 times.

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 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 (see z/OS Unicode Services User's Guide and Reference). 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 into a CICS TS 3.1 region, you must apply APAR PK23547 to achieve the minimum runtime level of code to install the web service binding file.

CHAR-VARYING={**NO**|**NULL**|**COLLAPSE**|**BINARY**}

Specifies how character fields in the language structure are mapped when the mapping level is 1.2 or higher. 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. You can select these options:

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. NO is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping levels 2.0 and earlier.

This value does not apply to Enterprise and Other PL/I language structures.

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. NULL is the default value for the **CHAR-VARYING** parameter for C/C++.

This value does not apply to Enterprise and Other PL/I language structures.

COLLAPSE
Character fields are mapped to an <xsd:string>. Trailing white space in the field is not included in the SOAP message. The inbound SOAP message is parsed to remove all leading, trailing, and embedded white space. COLLAPSE is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping level 2.1 onwards.

For more information about variable-length values and white space, see Support for variable-length values and white space.

BINARY
Character fields are mapped to an <xsd:base64binary> and are processed as fixed-length fields. The BINARY value on the **CHAR-VARYING** parameter is available only at mapping levels 2.1 and onwards.

CHAR-OCCURS={**STRING**|**ARRAY**}

Specifies how character arrays in the language structure are mapped when the mapping level is 4.0 or higher. For example, `PIC X OCCURS 20`. This parameter is only for use by the COBOL language.

ARRAY
Character arrays are mapped to an XML array. This means that every character is mapped as an individual XML element. This is also the behaviour at mapping levels 3.0 and earlier.

STRING
Character arrays are mapped to an XML string. This means that the entire COBOL array is mapped as a single XML element.

CHAR-USAGE={**NATIONAL**|**DBCS**}

In COBOL, the national data type, PIC N, can be used for UTF-16 or DBCS data. This setting is controlled by the NSYMBOL compiler option. You must set the **CHAR-USAGE** parameter on the assistant to the same value as the NSYMBOL compiler option to ensure that the data is handled appropriately. This is typically set to CHAR-USAGE=NATIONAL when you use UTF-16.

DBCS Data from PIC (*n*) fields is treated as UTF-16 encoded data.

NATIONAL
Data from PIC (*n*) fields is treated as DBCS encoded data.

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.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**UNUSED**|**PACKED15**}
Specifies if potential ABSTIME fields in the high-level language structure are mapped as timestamps:

PACKED15
Packed decimal fields of length 15 (8 bytes) are treated as CICS ABSTIME fields, and mapped as timestamps.

UNUSED
Packed decimal fields of length 15 (8 bytes) are not treated as timestamps.

You can set this parameter at a mapping level of 3.0.

HTTPPROXY={*domain name*:*port number*|*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 DFHLS2WS uses a proxy server to access the internet, specify the domain name or IP address and the port number of the proxy server. For example:

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

In other cases, this parameter is not required.

HTTPPROXY-PASSWORD=*value*
Specifies the HTTP proxy password that must be used with **HTTPPROXY-USERNAME** if the system on which you are running DFHLS2WS uses a HTTP proxy server to access the Internet, and the HTTP proxy server uses basic authentication. You can use this parameter only when you also specify **HTTPPROXY**.

HTTPPROXY-USERNAME=*value*
Specifies the HTTP proxy username that must be used with **HTTPPROXY-PASSWORD** if the system on which you are running DFHLS2WS uses a HTTP proxy server to access the Internet, and the HTTP proxy server uses basic authentication. You can use this parameter only 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 z/OS UNIX 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.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHLS2WS.

MAPPING-LEVEL={**1.0**|**1.1**|**1.2**|**2.0**|**2.1**|**2.2**|**3.0**|**4.0**}
Specifies the level of mapping that DFHLS2WS uses when generating the web service binding file and web service description. You can select these options:

1.0 This mapping level is the default. It indicates that the web service binding file is generated using CICS TS 3.1 mapping levels.

1.1 Use this mapping to regenerate a binding file at this specific level.

1.2 At this mapping level, you can use the **CHAR-VARYING** parameter to control how character arrays are processed at run time. VARYING and VARYINGZ arrays are also supported in PL/I.

2.0 Use this mapping level in a CICS TS 3.2 region or later to take advantage of the enhancements to the mapping between the language structure and web services binding file.

2.1 Use this mapping level with a CICS TS 3.2 region that has APAR PK59794 applied or with any region later than CICS TS 3.2. At this mapping level you can take advantage of the new values for the **CHAR-VARYING** parameter, `COLLAPSE` and `BINARY`. FILLER fields in COBOL and * fields in PL/I are systematically ignored at this mapping level, the fields do not appear in the generated WSDL document, and an appropriate gap is left in the data structures at run time.

2.2 Use this mapping level with a CICS TS 3.2 region that has APAR PK69738 applied or with any region later than CICS TS 3.2 to take advantage of mapping enhancements when using DFHWS2LS.

3.0 Use this mapping level with a CICS TS 4.1 region. At this mapping level you can create a web service from an application that uses many containers in its interface by setting the **REQUEST-CHANNEL** and **RESPONSE-CHANNEL** parameters. You can also map dateTime fields to XML timestamps by setting the **DATETIME** parameter.

4.0 Use this mapping level with a CICS TS 5.2 region. At this mapping level you can use COBOL `OCCURS DEPENDING ON` fields and the **CHAR-OCCURS** parameter.

For more information about mapping levels, see Mapping levels for the CICS assistants

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}
Specifies the minimum CICS runtime environment into which the web service binding file can be deployed. If you select a level that does not match the other parameters that you have specified, you receive an error message. You can select these options:

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. Some parameters are not available at this runtime level.

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 can use a mapping level of 1.1 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

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 a mapping level of 1.2 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.0 The generated web service binding file deploys successfully into a CICS TS 3.2 region or later. You can use a mapping level of 2.0 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.1 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK59794 applied or into any region later than CICS TS 3.2. You can use a mapping level of 2.1 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

2.2 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK69738 applied or into any region later than CICS TS 3.2. With this runtime level, you can use a mapping level of 2.2 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

3.0 The generated web service binding file deploys successfully into a CICS TS 4.1 region or later. With this runtime level, you can use a mapping level of 3.0 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a

mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

OPERATION-NAME=*value*
Specifies the operation name that is used in the generated WSDL document. If no value is supplied, then a default name is generated using the value of the **PGMNAME** parameter followed by value **operation**.

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, the z/OS UNIX System Services code page is used. For example, you might 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.

- At mapping levels earlier than 3.0, the channel can contain only one container, which is used for both input and output. Use the **CONTID** parameter to specify the name of the container. The default name is DFHWS-DATA.
- At mapping level 3.0, the channel can contain multiple containers. Use the **REQUEST-CHANNEL** and **RESPONSE-CHANNEL** parameters. Do not specify **PDSLIB**, **REQMEM**, or **RESPMEM**.

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

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of the CICS PROGRAM resource for the target application program that will be exposed as a web service. The CICS web service support will link to this program.

REQMEM=*value*
Specifies the name of the partitioned data set member that 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.

REQUEST-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when receiving a SOAP message from a web service requester. The channel description is an XML document that must conform to the CICS-supplied channel schema.

You can use this parameter at mapping level 3.0 only.

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 that 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.

Omit this parameter if no response is involved; that is, for one-way messages.

RESPONSE-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when sending a SOAP response message to a web service requester. The channel description is an XML document that must conform to the CICS-supplied channel schema.

You can use this parameter at mapping level 3.0 only.

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.

SOAPVER={**1.1**|**1.2**|**ALL**}
Specifies the SOAP level to use in the generated web service description. This parameter is available only when the **MINIMUM-RUNTIME-LEVEL** is set to 2.0 or higher.

1.1 The SOAP 1.1 protocol is used as the binding for the web service description.

1.2 The SOAP 1.2 protocol is used as the binding for the web service description.

ALL Both the SOAP 1.1 or 1.2 protocol can be used as the binding for the web service description.

If you do not specify a value for this parameter, the default value depends on the version of WSDL that you want to create:

- If you require only WSDL 1.1, the SOAP 1.1 binding is used.
- If you require only WSDL 2.0, the SOAP 1.2 binding is used.
- If you require both WSDL 1.1 and WSDL 2.0, both SOAP 1.1 and 1.2 bindings are used for each web service description.

SSL-KEYSTORE=*value*
This optional parameter specifies the fully qualified location of the key store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-KEYPWD=*value*
This optional parameter specifies the password for the key store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTSTORE=*value*
This optional parameter specifies the fully qualified location of the trust store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTPWD=*value*
This optional parameter specifies the password for the trust store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

STRUCTURE=(*request*,*response*)
For C and C++ only, specifies the names of the high-level structures contained in the partitioned data set members that are specified in the **REQMEM** and **RESPMEM** parameters:

request
Specifies the name of the high-level structure that contains 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 sync point.

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

YES The remote web service can issue a sync point. 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- to 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*
This parameter specifies the relative or absolute 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- to 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 z/OS UNIX name of the web service binding file. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is .wsbind.

WSDL=*value*
The fully qualified z/OS UNIX name of the file into which the web service description is written. The web service description conforms to the WSDL 1.1 specification. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is .wsdl.

WSDL_1.1=*value*
The fully qualified z/OS UNIX name of the file into which the web service description is written. The web service description conforms to the WSDL 1.1 specification. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is .wsdl. This parameter produces the same result as the **WSDL** parameter, so you can specify only one or the other.

WSDL_2.0=*value*
The fully qualified z/OS UNIX name of the file into which the web service description is written. The web service description conforms to the WSDL 2.0 specification. DFHLS2WS creates the file, but not the directory structure, if it does not already exist. The file extension is .wsdl. This parameter can be used with the **WSDL** or **WSDL_1.1** parameters. It is available only when the **MINIMUM-RUNTIME-LEVEL** is set to 2.0 or higher.

WSDLCP={**LOCAL**|**UTF-8**|**EBCDIC-CP-US**}
Specifies the code page that is used to generate the WSDL document.

LOCAL
Specifies that the WSDL document is generated using the local code page and no encoding tag is generated in the WSDL document.

UTF-8 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.

EBCDIC-CP-US
This value specifies that the WSDL document is generated using the US EBCDIC code page. An encoding tag is generated in the WSDL document.

WSDL-NAMESPACE=*value*
Specifies the namespace for CICS to use in the generated WSDL document.

If you do not specify this parameter, CICS generates a namespace automatically.

WSRR-CUSTOM-*PropertyName=value*
Use this optional parameter to add customized metadata to the WSDL document in the WSRR. The WSRR-CUSTOM-*PropertyName=value* pairs are added into the WSDL document and appear in WSRR without the WSSR-CUSTOM prefix.

You can specify a maximum of 255 custom *PropertyName=value* pairs. Avoid duplicate and blank *PropertyName=value* pairs.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-DESCRIPTION=*value*
Use this optional parameter to specify the metadata that describes the WSDL document being published.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-ENCODING=*value*
Use this optional parameter to specify the character set encoding of the WSDL document. If the **WSRR-ENCODING** parameter is not specified, WSRR uses the value specified in the WSDL document.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-LOCATION=*value*
Use this optional parameter to specify the URI that identifies the location of the WSDL document. If this parameter is not specified, the URI defaults to the filename specified in the **WSDL** parameter. For example, if the value of the **WSDL** parameter is `wsrr/example.wsdl`, the value of the **WSRR-LOCATION** parameter defaults to `example.wsdl`.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-PASSWORD=*value*
Use this optional parameter if you must enter a password to access WSRR.

If the **WSRR-USERNAME** parameter is specified, you must also specify this parameter.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-SERVER={*domain name*:*port number*|*IP address*:*port number*}
Use this parameter to specify the location of the IBM WebSphere Service Registry and Repository (WSRR) server. If this parameter is specified, WSRR parameter validation is used.

WSRR-USERNAME=*value*
Use this optional parameter if you are required to specify a user name to access WSRR. This user name is used by WSRR to set the owner property.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-VERSION={**1**|*value*}
Use this parameter to set the version property of the WSDL document in WSRR.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

Other information

- The user ID under which DFHLS2SC runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

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/wsbind/example.log
MINIMUM-RUNTIME-LEVEL=2.1
MAPPING-LEVEL=2.1
CHAR-VARYING=COLLAPSE
PGMNAME=DFH0XCMN
URI=http://myserver.example.org:8080/exampleApp/example
PGMINT=COMMAREA
SOAPVER=1.1
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
WSDL=/u/exampleapp/wsdl/example.wsdl
WSDL_2.0=/u/exampleapp/wsdl/example_20.wsdl
WSDLCP=LOCAL
WSDL-NAMESPACE=http://mywsdlnamespace
/*
```

DFHWS2LS: WSDL to high-level language conversion

The DFHWS2LS procedure 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.

The job control statements for DFHWS2LS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

Job control statements for DFHWS2LS

JOB Starts the job.

EXEC Specifies the procedure name (DFHWS2LS).

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 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/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 z/OS UNIX directory path used on other parameters. The default is the empty string.

Typically, you do not 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 z/OS UNIX that DFHWS2LS uses as a temporary workspace. 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 file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*.

Typically, you do not 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 at run time:

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 as follows:

/tmp/WS2LS.in

/tmp/WS2LS.out

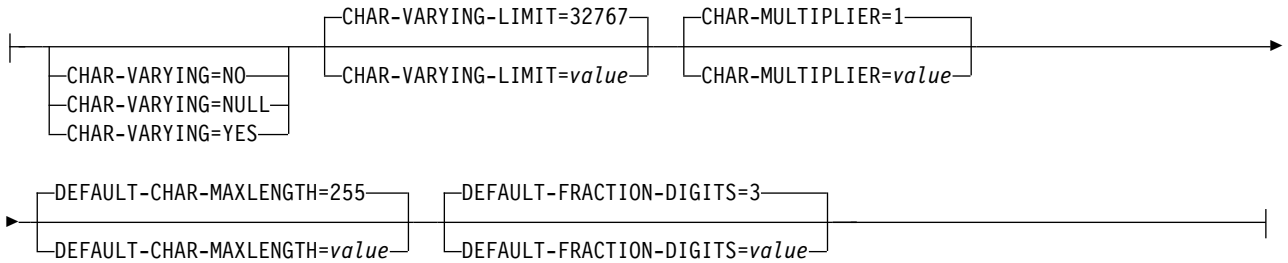
/tmp/WS2LS.err

Important: DFHWS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHWS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job overwriting the workspace files while another job is using them, leading to unpredictable failures.

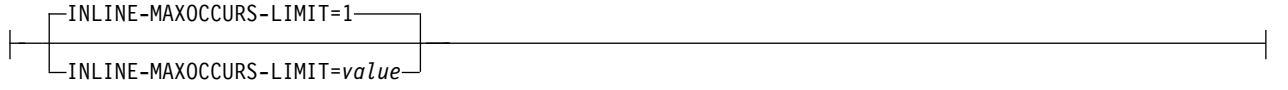
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of the job.

Input parameters for DFHWS2LS

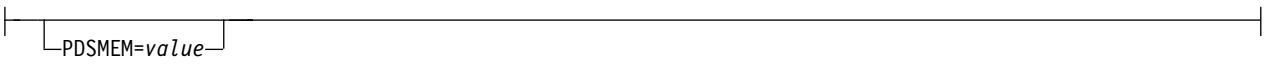
Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):

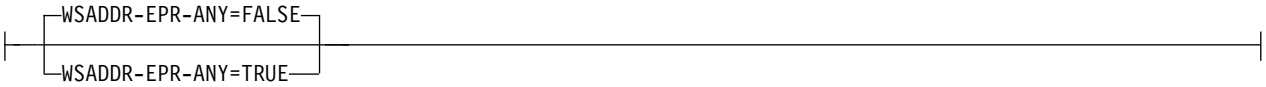


Advanced data mapping (mapping level 2.2 and higher):



Advanced data mapping (runtime level 2.1 and higher):

Advanced data mapping (runtime level 3.0 and higher):



Notes:

1 Each of the WSRR parameters that can be specified when the **WSRR-SERVER** parameter is set can be specified only once.

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 must 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.
- A comma in the last character position of the line is an optional line separator, and is ignored.

Parameter descriptions

BINDING=*value*
: If the web service description contains more than one <wsdl: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 <wsdl: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 (see z/OS

Unicode Services User's Guide and Reference). 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 into a CICS TS 3.1 region, you must apply APAR PK23547 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 or later. The *value* of this parameter can be a positive integer in the range of 1 - 2,147,483,647. 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 might 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.

When you set **CCSID=1200** (indicating UTF-16), the only valid values for **CHAR-MULTIPLIER** are 2 or 4. When you use UTF-16, the default value is 2. Use **CHAR-MULTIPLIER=2** when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use **CHAR-MULTIPLIER=4** when you expect application data to contain characters that require 2 UTF-16 encoding units.

Note: Setting **CHAR-MULTIPLIER** to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger **CHAR-MULTIPLIER** value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.

CHAR-VARYING={**NO**|**NULL**|**YES**}
Specifies how variable-length character data is mapped when the mapping level is 1.2 or higher. Variable-length binary data types are always mapped to either a container or a varying structure. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

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 two related elements: data-length and the data.

CHAR-VARYING-LIMIT={**32767**|*value*}
Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure when the mapping level is 1.2 or higher. If the character or binary 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 32,767 bytes.

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.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**PACKED15**|**STRING**}
Specifies how <xsd:dateTime> elements are mapped to the language structure.

PACKED15
The default is that any <xsd:dateTime> element is processed as a timestamp and is mapped to CICS ABSTIME format.

STRING
The <xsd:dateTime> element is processed as text.

DEFAULT-CHAR-MAXLENGTH={**255**|*value*}
Specifies the default array length of character data in characters for mappings where no length is implied in the web service description document, when the mapping level is 1.2 or higher. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647.

DEFAULT-FRACTION-DIGITS={**3**|*value*}
Specifies the default number of fraction digits to use on an XML decimal schema type. The default is 3. For COBOL, the valid range is 0-17, or 0-30 if parameter **WIDE-COMP3** is being used. For C or PLI the valid range is 0-30.

HTTPPROXY={*domain name*:*port number*|*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 the port number of the proxy server. For example:

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

In other cases, this parameter is not required.

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

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

INLINE-MAXOCCURS-LIMIT={**1**|*value*}
Specifies whether or not inline variable repeating content is used based on the

maxOccurs attribute. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.

The **INLINE-MAXOCCURS-LIMIT** parameter is available only at mapping level 2.1 onwards. The value of **INLINE-MAXOCCURS-LIMIT** can be a positive integer in the range of 0 - 32,767. A value of 0 indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the *value* of the maxOccurs attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise inline mapping is used.

When deciding if you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.

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 z/OS UNIX 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.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHWS2LS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHWS2LS uses when generating the web service binding file and language structure. You can select these options:

1.0 The web service binding file and language structure are generated using CICS TS 3.1 mapping levels.

1.1 XML attributes and <list> and <union> data types are mapped to the language structure. Character and binary data that have a maximum length of more than 32,767 bytes are mapped to a container. The container name is created in the language structure.

1.2 Use the **CHAR-VARYING** and **CHAR-VARYING-LIMIT** parameters 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 have a maximum length of less than 32,768 bytes are mapped to a VARYING structure for all languages except C++, where character data is mapped to a null-terminated string.

2.0 Use this mapping level in a CICS TS 3.2 region or later to take advantage of the enhancements to the mapping between the language structure and web services binding file.

2.1 Use this mapping level with a CICS TS 3.2 region that has APAR PK59794 applied, or any region later than CICS TS 3.2 for <xsd:any> and xsd:anyType support, the option to map variably repeating content inline with the **INLINE-MAXOCCURS-LIMIT** parameter, and support for minOccurs="0" on <xsd:sequence>, <xsd:choice>, and <xsd:all>.

2.2 Use this mapping level with a CICS TS 3.2 region that has APAR PK69738 applied or with any region later than CICS TS 3.2. It provides the following support:

- Elements with fixed values
- Enhanced support for <xsd:choice> elements
- Abstract data types
- Abstract elements
- Substitution groups

3.0 Use this mapping level with a CICS TS 4.1 or later region. At this mapping level you can transform timestamps to CICS ABSTIME format.

4.0 Use this mapping level with a CICS TS 5.2 region when you want to use UTF-16.

For more information about mapping levels, see Mapping levels for the CICS assistants.

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|LESS-DUP-NAMES|NO-ARRAY-NAME-INDEXING}

Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

Note: Any of the sub options may be used in a comma delimited list. The options are not mutually exclusive, they are combinatorial and unordered.

LESS-DUP-NAMES

This parameter generates non-structural structure field names with `_value` at the end of the name to enable direct referencing to the field. For example, in the following PLI language structure, when MAPPING-OVERRIDES=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with `_value`:

```
09 streetName,
    12 streetName                    CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

The resulting structure is as follows:

```
09 streetName,
    12 streetName_value              CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

SAME-AS-MAPPING-LEVEL
This parameter generates language structures in the same style as the mapping level. This is the default.

UNDERSCORES-AS-HYPHENS
For COBOL only. This parameter converts any underscores in the WSDL document to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure they are unique. For more information, see XML schema to COBOL mapping in Developing applications.

NO-ARRAY-NAME-INDEXING
For Enterprise PL/I only. Ensures that the field names within an array are unique only within the scope of the higher level structure.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}
Specifies the minimum CICS runtime environment into which the web service binding file can be deployed. If you select a level that does not match the other parameters that you have specified, you receive an error message. You can select these options:

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. Some parameters are not available at this runtime level.

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 can use a mapping level of 1.1 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

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 a mapping level of 1.2 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.0 The generated web service binding file deploys successfully into a CICS TS 3.2 region or later. You can use a mapping level of 2.0 or earlier for the MAPPING-LEVEL parameter. Some parameters are not available at this runtime level.

2.1 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK59794 applied or into any region later than CICS TS 3.2. You can use a mapping level of 2.1 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

2.2 The generated web service binding file deploys successfully into a CICS TS 3.2 region that has APAR PK69738 applied or into any region later than CICS TS 3.2. With this runtime level, you can use a mapping level of 2.2 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

3.0 The generated web service binding file deploys successfully into a CICS TS 4.1 region or later. With this runtime level, you can use a mapping level of 3.0 or earlier for the **MAPPING-LEVEL** parameter. Some parameters are not available at this runtime level.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

NAME-TRUNCATION={LEFT|RIGHT}
Specifies whether XML element names are truncated from the left or the right. The CICS web services assistant truncates XML element names to the appropriate length for the high-level language specified; by default names are truncated from the right.

OPERATIONS=*value*
For web service requester applications, specifies a subset of valid <wsdl:Operation> elements from the web service description that is used to generate the web service binding file. Each <wsdl:Operation> element is separated by a space; the list can span more than one line if necessary. You can use this parameter for both WSDL 1.1 and WSDL 2.0 documents.

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, the z/OS UNIX System Services code page is used. For example, you might specify **PDSCP**=037.

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.

PDSMEM=*value*
Specifies a 1- to 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 abstract data types. It generates the member name by appending a 2-digit number to the prefix.

Use this parameter at a mapping level of 2.2 or higher for naming the language structures associated with abstract data types. If the **PDSMEM** parameter is omitted, language structures for abstract data types are named using the value in the **REQMEM** parameter.

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.

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of a CICS PROGRAM resource.

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

When DFHWS2LS is used to generate a web service binding file that will be used in a service requester, omit this parameter.

REQMEM=*value*
Specifies a 1- to 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 2-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- to 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 2-digit number to the prefix.

Omit this parameter if no response is involved; that is, for one-way messages.

SSL-KEYSTORE=*value*
This optional parameter specifies the fully qualified location of the key store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-KEYPWD=*value*
This optional parameter specifies the password for the key store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTSTORE=*value*
This optional parameter specifies the fully qualified location of the trust store file.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

SSL-TRUSTPWD=*value*
This optional parameter specifies the password for the trust store.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network to an IBM WebSphere Service Registry and Repository (WSRR).

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 sync point.

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

YES The remote web service can issue a sync point. 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- to 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 uses 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. CICS does not generate a URIMAP resource for a service

requester. You can define your own URIMAP resource for service requesters to use when they make client requests to the URI of the target web service. When a service requester issues the **INVOKE SERVICE** command, CICS uses the `soap:address` location from the `wsdl:port` specified in the web service description if present. You can override that and specify a different URI using the `URIMAP` or `URI` options on the **INVOKE SERVICE** command.

USERID=*id*
In a service provider, this parameter specifies a 1- to 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 $ @ #`

WIDE-COMP3={**NO**|**YES**}
For COBOL only. Controls the maximum size of the packed decimal variable length in the COBOL language structure.

NO DFHWS2LS limits the packed decimal variable length to 18 when generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.

YES DFHWS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

WSADDR-EPR-ANY={**TRUE**|**FALSE**}
Specifies whether CICS transforms a WS-Addressing endpoint reference (EPR) into its components parts in the language structures or treats the EPR as an <xsd:any> type. Treating the EPR as an <xsd:any> type means that the **WSACONTEXT BUILD** API can use the EPR XML directly.

FALSE
DFHWS2LS behaves typically, transforming the XML to a high-level language structure.

TRUE Setting this option to `TRUE` means that at run time CICS treats the whole EPR as an <xsd:any> type and places the EPR XML into a container that can be referenced by the application. The application can use the EPR XML with the **WSACONTEXT BUILD** API to construct an EPR in the addressing context.

This parameter is available only at runtime level 3.0 onwards.

WSBIND=*value*
The fully qualified z/OS UNIX name of the web service binding file. DFHWS2LS creates the file, but not the directory structure, if it does not already exist. The file extension defaults to `.wsbind`.

WSDL=*value*
The fully qualified z/OS UNIX name of the file that contains the web service description. If you are using WSRR to retrieve the WSDL document, this parameter specifies the location on the file system to which a local copy of the WSDL document will be written.

WSDL-SERVICE=*value*
Specifies the `wsdl:Service` element that is used when the web service

description contains more than one Service element for a Binding element. If you specify a value for the **BINDING** parameter, the Service element that you specify for this parameter must be consistent with the specified Binding element. You can use this parameter with either WSDL 1.1 or WSDL 2.0 documents.

WSRR-NAME=*value*
Specifies the name of the WSDL document to retrieve from WSRR. Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-NAMESPACE=*value*
Specifies the namespace of the WSDL document to retrieve from WSRR. You can optionally use this parameter when the **WSRR-SERVER** parameter is specified to fully qualify the WSDL document name specified in the **WSRR-NAME** parameter.

WSRR-PASSWORD=*value*
Use this optional parameter if you must enter a password to access WSRR.

If the **WSRR-USERNAME** parameter is specified, you must also specify this parameter.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-SERVER={*domain name*:*port number*|*IP address*:*port number*}
Use this parameter to specify the location of the IBM WebSphere Service Registry and Repository (WSRR) server. If this parameter is specified, WSRR parameter validation is used.

WSRR-USERNAME=*value*
Use this optional parameter if you are required to specify a user name to access WSRR. This user name is used by WSRR to set the owner property.

Use this parameter only when the **WSRR-SERVER** parameter is specified.

WSRR-VERSION=*value*
Specifies the version of the WSDL document to retrieve from WSRR. You can use this parameter only when the **WSRR-SERVER** parameter is specified.

XML-ONLY={**TRUE**|**FALSE**}
Specifies whether or not CICS transforms the XML in the SOAP message to application data. Use the **XML-ONLY** parameter to write web service applications that process the XML themselves.

TRUE CICS does not perform any transformations to the XML. The service requester or provider application must work with the contents of the DFHWS-BODY container directly to map data between XML and the high-level language.

FALSE
CICS transforms the XML to a high-level language.

This parameter is available only at runtime level 2.1 onwards.

Other information

- The user ID under which DFHLS2SC runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.

- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement. For more information, see z/OS UNIX System Services User's Guide.

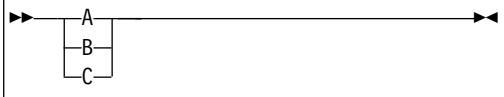
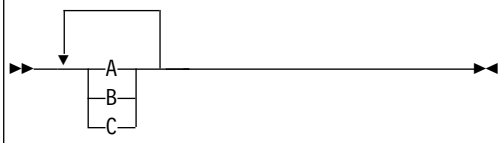
Example

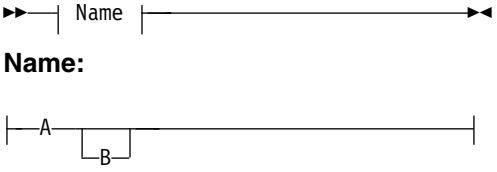
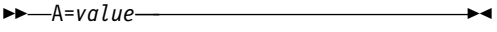
```
//WS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHWS2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=3.0
MAPPING-OVERRIDES=UNDERSCORES-AS-HYPHENS
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
WSDL=/u/exampleapp/wsdl/example.wsdl
/*
```

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. *value* denotes a variable, for which you should specify an appropriate value.

Mapping levels for the CICS assistants

A mapping is the set of rules that specifies how information is converted between language structures and XML schemas. To benefit from the most sophisticated mappings available, you are recommended to set the **MAPPING-LEVEL** parameter in the CICS assistants to the latest level.

Each level of mapping inherits the function of the previous mapping, with the highest level of mapping offering the best capabilities available. The highest mapping level provides more control over data conversion at run time and removes restrictions on support for certain data types and XML elements.

You can set the **MAPPING-LEVEL** parameter to an earlier level if you want to redeploy applications that were previously enabled at that level.

Mapping level 3.0

Mapping level 3.0 is compatible with a CICS TS 4.1 region.

This mapping level provides the following support:

- DFHSC2LS and DFHWS2LS map `xsd:dateTime` data types to CICS ASKTIME format.
- DFHLS2WS can generate a WSDL document and web service binding from an application that uses many containers rather than just one container.

- Tolerating truncated data that is described by a fixed length data structure. You can set this behavior by using the **DATA-TRUNCATION** parameter on the CICS assistants.

Mapping level 2.2 and higher

Mapping level 2.2 is compatible with a CICS TS 3.2 region, with APAR PK69738 applied, and higher.

At mapping level 2.2 and higher, DFHSC2LS and DFHWS2LS support the following XML mappings:

- Fixed values for elements
- Substitution groups
- Abstract data types
- XML schema <sequence> elements can nest inside <choice> elements

DFHSC2LS and DFHWS2LS provide enhanced support for the following XML mappings:

- Abstract elements
- XML schema <choice> elements

Mapping level 2.1 and higher

Mapping level 2.1 is compatible with a CICS TS 3.2 region, with APAR PK59794 applied, and higher.

This mapping level includes greater control over the way variable content is handled with the new **INLINE-MAXOCCURS-LIMIT** parameter and new values on the **CHAR-VARYING** parameter.

At mapping level 2.1 and higher, DFHSC2LS and DFHWS2LS offer the following new and improved support for XML mappings:

- The XML schema <any> element
- The `xsd:anyType` type
- Toleration of abstract elements
- The **INLINE-MAXOCCURS-LIMIT** parameter
- The `minOccurs` attribute

The **INLINE-MAXOCCURS-LIMIT** parameter specifies whether variably repeating lists are mapped inline. For more information on mapping variably repeating content inline, see Variable arrays of elements.

Support for the `minOccurs` attribute has been enhanced on the XML schema <sequence>, <choice>, and <all> elements. If `minOccurs="0"`, the CICS assistant treats these element as though the `minOccurs="0"` attribute is also an attribute of all its child elements.

At mapping level 2.1 and higher, DFHLS2SC and DFHLS2WS support the following XML mappings:

- FILLER fields in COBOL and PL/I are ignored
- A value of COLLAPSE for the **CHAR-VARYING** parameter
- A value of BINARY for the **CHAR-VARYING** parameter

FILLER fields in COBOL and PL/I are ignored; they do not appear in the generated XML schema and an appropriate gap is left in the data structures at run time.

`COLLAPSE` causes CICS to ignore trailing spaces in text fields.

`BINARY` provides support for binary fields. This value is useful when converting COBOL into an XML schema. This option is available only on SBCS character arrays and allows the array to be mapped to fixed-length `xsd:base64Binary` fields rather than to `xsd:string` fields.

Mapping level 1.2 and higher

Mapping level 1.2 is compatible with a CICS TS 3.1 region and higher.

Greater control is available over the way character and binary data are transformed at run time with these additional parameters on the batch tools:

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**
- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

If you decide to use the **CHAR-MULTIPLIER** parameter in DFHSC2LS or DFHWS2LS, note that the following rules apply after the value of this parameter is used to calculate the amount of space required for character data.

- DFHSC2LS and DFHWS2LS provide these mappings:
  - Variable-length character data types that have a maximum length of more than 32 767 bytes map 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.
  - Variable-length character data types that have a maximum length of less than 32 768 bytes map 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 the way that variable-length character data is mapped.
  - Variable-length binary data that has a maximum length of less than 32 768 bytes maps 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.

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 or DFHSC2LS.

DFHLS2SC and DFHLS2WS provide these mappings:

- 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 the way that variable-length character data is handled at run time for all languages except PL/I.

- Base64Binary data types map 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.

Mapping level 1.1 and higher

Mapping level 1.1 is compatible with a CICS TS 3.1 region and higher.

This mapping level provides improved mapping of XML character and binary data types, in particular when mapping data of variable length that has `maxLength` and `minLength` attributes defined with different values in the XML schema. Data is handled in the following ways:

- Character and binary data types that have a fixed length that is greater than 16 MB 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.

  Because 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 XML schema or web service description. If the length of the data is significant, you can either write your application to check it or turn validation on in the CICS region. Both SOAP and XML validation have a significant performance impact.
- XML schema <list> and <union> data types map to character fields.
- Schema-defined XML attributes are mapped rather than ignored. A maximum of 255 attributes is allowed for each XML element. See Support for XML attributes for further information.
- The `xsi:nil` attribute is supported. See Support for XML attributes for further information.

Mapping level 1.1 only

Mapping level 1.1 is compatible with a CICS TS 3.1 region and higher.

This mapping level provides improved mapping of XML character and binary data types, in particular when mapping data of variable length that has `maxLength` and `minLength` attributes 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 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 of less than 16 MB 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.

- CICS encodes and decodes data in the hexBinary format but not in base64Binary format. 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.
  - ceil(x) is the smallest integer greater than or equal to x.

  If the length of z is greater than 24 566 bytes, the resulting language structure fails to compile. If you have base64Binary data that is greater than 24 566 bytes, you are recommended to use a mapping level of 1.2. With mapping level 1.2, you can map the base64Binary data to a container instead of using a field in the language structure.

Mapping level 1.0 only

Mapping level 1.0 is compatible with a CICS TS 3.1 region and higher.

Note the following limitations, which have been modified in later mapping levels:

- DFHSC2LS and DFHWS2LS map character and binary data types in the XML schema to fixed-length fields in the language structure. Look at this partial XML schema:

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

  That partial XML schema appears in a COBOL language structure like this:

  ```
  15 example     PIC X(33000)
  ```
- CICS encodes and decodes data in the hexBinary format but not in base64Binary format. DFHSC2LS and DFHWS2LS map Base64Binary data to a fixed-length character field, the contents of which must be encoded or decoded by the application program.
- DFHSC2LS and DFHWS2LS ignore XML attributes during processing.
- DFHLS2SC and DFHLS2WS interpret character and binary fields in the language structure as fixed-length fields and map those 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 insufficient data is available.

High-level language and XML schema mapping

Use the CICS assistants to generate mappings between high-level language structures and XML schemas or WSDL documents. The CICS assistants also generate XML schemas or WSDL documents from high-level language data structures, or vice-versa.

Utility programs DFHSC2LS and DFHLS2SC are collectively known as the CICS XML assistant. Utility programs DFHWS2LS and DFHLS2WS are collectively known as the CICS Web services assistant.

- DFHLS2SC and DFHLS2WS map high-level language structures to XML schemas and WSDL documents respectively.
- DFHSC2LS and DFHWS2LS map XML schemas and WSDL documents to high-level language structures.

The two mappings are not symmetrical:

- If you process a language data structure with DFHLS2SC or DFHLS2WS and then process the resulting XML schema or WSDL document with the complementary utility program (DFHSC2LS or DFHWS2LS respectively), do not expect the final data structure to be the same as the original. However, the final data structure is logically equivalent to the original.
- If you process an XML schema or WSDL document with DFHSC2LS or DFHWS2LS and then process the resulting language structure with the complementary utility program (DFHLS2SC or DFHLS2WS respectively), do not expect the XML schema in the final XML schema or WSDL document to be the same as the original.
- In some cases, DFHSC2LS and DFHWS2LS generate language structures that are not supported by DFHLS2SC and DFHLS2WS.

You must code language structures processed by DFHLS2SC and DFHLS2WS according to the rules of the language, as implemented in the language compilers that CICS supports.

Data mapping limitations when using the CICS assistants

CICS supports bidirectional data mappings between high-level language structures and XML schemas or WSDL documents that conform to WSDL version 1.1 or 2.0, with certain limitations. These limitations apply only to the DFHWS2LS and DFHSC2LS tools and vary according to the mapping level.

Limitations at all mapping levels

- Only SOAP bindings that use literal encoding are supported. Therefore, you must set the `use` attribute to a value of `literal`; `use="encoded"` is not supported.
- Data type definitions must be encoded using the XML Schema Definition language (XSD). In the schema, data types used in the SOAP message must be explicitly declared.
- The length of some keywords in the web services description is limited. For example, operation, binding, and part names are limited to 255 characters. In some cases, the maximum operation name length might be slightly shorter.
- 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.
- DFHWS2LS and DFHSC2LS support only a single <xsd:any> element in a particular scope. For example, the following schema fragment is not supported:

  ```
  <xsd:sequence>
   <xsd:any/>
   <xsd:any/>
  </xsd:sequence>
  ```

  Here, `<xsd:any>` can specify `minOccurs` and `maxOccurs` if required. For example, the following schema fragment is supported:

  ```
  <xsd:sequence>
   <xsd:any minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
  ```

- Cyclic references are not supported. For example, where type A contains type B which, in turn, contains type A.
- Recurrence is not supported in group elements, such as <xsd:choice>, <xsd:sequence>, <xsd:group>, or <xsd:all> elements. For example, the following schema fragment is not supported:

```
<xsd:choice maxOccurs="2">
   <xsd:element name="name1" type="string"/>
</xsd:choice>
```

The exception is at mapping level 2.1 and higher, where `maxOccurs="1"` and `minOccurs="0"` are supported on these elements.

- DFHSC2LS and DFHWS2LS do not support data types and elements in the SOAP message that are derived from the declared data types and elements in the XML schema either from the `xsi:type` attribute or from a substitution group, except at mapping level 2.2 and higher if the parent element or type is defined as abstract.
- Embedded <xsd:sequence> and <xsd:group> elements inside an <xsd:choice> element are not supported prior to mapping level 2.2. Embedded <xsd:choice> and <xsd:all> elements inside an <xsd:choice> element are never supported.

Improved support at mapping level 1.1 and higher

When the mapping level is 1.1 or higher, DFHWS2LS provides support for the following XML elements and element type:

- The <xsd:list> element.
- The <xsd:union> element.
- The `xsd:anySimpleType` type.
- The <xsd:attribute> element. At mapping level 1.0 this element is ignored.

Improved support at mapping level 2.1 and higher

When the mapping level is 2.1 or higher, DFHWS2LS supports the following XML elements and element attributes:

- The <xsd:any> element.
- The `xsd:anyType` type.
- Abstract elements. In earlier mapping levels, abstract elements are supported only as nonterminal types in an inheritance hierarchy.
- The `maxOccurs` and `minOccurs` attributes on the <xsd:all>, <xsd:choice>, and <xsd:sequence> elements, only when `maxOccurs="1"` and `minOccurs="0"`.
- "FILLER" fields in COBOL and "*" fields in PL/I are suppressed. The fields do not appear in the generated WSDL and an appropriate gap is left in the data structures at run time.

Improved support at mapping level 2.2 and higher

When the mapping level is 2.2 or higher, DFHSC2LS and DFHWS2LS provide improved support for the <xsd:choice> element, supporting a maximum of 255 options in the <xsd:choice> element. For more information on <xsd:choice> support, see “Support for `<xsd:choice>`” on page 120.

At mapping level 2.2 and higher, the CICS assistants support the following XML mappings:

- Substitution groups
- Fixed values for elements
- Abstract data types

Embedded <xsd:sequence> and <xsd:group> elements inside an <xsd:choice> element are supported at mapping level 2.2 and higher. For example, the following schema fragment is supported:

```
<xsd:choice>
   <xsd:element name="name1" type="string"/>
   <xsd:sequence/>
</xsd:choice>
```

If the parent element or type in the SOAP message is defined as abstract, DFHSC2LS and DFHWS2LS support data types and elements that are derived from the declared data types and elements in the XML schema.

Improved support at mapping level 3.0 and higher

When the mapping level is 3.0 or higher, the CICS assistants support the following mapping improvements:

- DFHSC2LS and DFHWS2LS map `xsd:dateTime` data types to CICS ASKTIME format.
- DFHLS2WS can generate a WSDL document and web service binding from an application that uses many containers rather than just one container.
- Tolerating truncated data that is described by a fixed length data structure. You can set this behavior by using the **DATA-TRUNCATION** parameter on the CICS assistants.

COBOL to XML schema mapping

The DFHLS2SC and DFHLS2WS utility programs support mappings between COBOL data structures and XML schema definitions.

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 uppercase characters are converted to lowercase.

   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`.

CICS 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 DFHLS2SC or 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 supported only when the mapping level specified in DFHLS2SC or DFHLS2WS is 1.2 or higher.
- SEPARATE CHARACTER is supported at a mapping level of 1.2 or higher 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 or higher:

  COMPUTATIONAL-1

  COMPUTATIONAL-2
- The only PICTURE characters supported for DISPLAY and COMPUTATIONAL-5 data description items are 9, S, and Z.
- The PICTURE characters supported for PACKED-DECIMAL data description items are 9, S, V, and Z.
- The only PICTURE characters supported for edited numeric data description items are 9 and Z.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an `xsd:string` and are processed as null-terminated strings.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to BINARY, character arrays are mapped to `xsd:base64Binary` and are processed as binary data.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to COLLAPSE, trailing white space is ignored for strings.

COBOL data description	Schema simpleType
PIC X(*n*) PIC A(*n*) PIC G(*n*) DISPLAY-1 PIC N(*n*)	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxlength value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> where *m* =*n*

COBOL data description	Schema simpleType
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:short"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC S9(*z*) DISPLAY where 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:int"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC S9(*z*) DISPLAY where 9 < *z*	<xsd:simpleType> <xsd:restriction base="xsd:long"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC 9(*z*) DISPLAY where 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.
PIC 9(*z*) DISPLAY where 9 < *z*	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> where *n* is the maximum value that can be represented by the pattern of '9' characters.

COBOL data description	Schema simpleType
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY where $n \leq 4$.	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY where $5 \leq n \leq 9$.	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY where $9 < n$.	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY where $n \leq 4$.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY where $5 \leq n \leq 9$.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY where $9 < n$.	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>
PIC S9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> where $p = m + n$.
PIC 9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> <xsd:minInclusive value="0"/> </xsd:restriction> </xsd:simpleType> where $p = m + n$.

COBOL data description	Schema simpleType
PIC S9(*m*) COMP-3 Supported at mapping level 3.0 when DATETIME=PACKED15	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> The format of the timestamp is CICS ABSTIME.
PIC S9(*m*)V9(*n*) DISPLAY Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> where $p = m + n$.
COMP-1 Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
COMP-2 Supported at mapping level 1.2 and higher	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

XML schema to COBOL mapping

The DFHSC2LS and DFHWS2LS utility programs support mappings between XML schema definitions and COBOL data structures.

The CICS assistants generate 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. You can use the **MAPPING-OVERRIDES** parameter to change the way other characters are handled. For example, if you set the value UNDERSCORES-AS-HYPHENS, any underscore in the XML is converted to a hyphen instead of an X. So monthly_total becomes monthly-total.

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 on an xsd:element 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 with different values.

   For more information, see "Variable arrays of elements" on page 112.

7. For attributes, the previous rules are applied to the element name. The prefix `attr-` is added to the element name, and is followed by `-value` or `-exist`. If the total length is longer than 28 characters, the element name is truncated. For more information, see "Support for XML attributes" on page 116.

   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 30 characters or less.

DFHSC2LS and DFHWS2LS map schema types to COBOL data description elements by using the specified mapping level according to the following table. 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 and an extra character is allocated for the null-terminator.
- 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

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anyType">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 2.0 and below:
	Not supported
	
	Mapping level 2.1:
	
	Supported

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anySimpletype">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 1.0:
	Not supported
	
	Mapping level 1.1 and higher:
	
	`PIC X(255)`

Schema simple type	COBOL data description

<xsd:simpleType>
  <xsd:restriction base="xsd:*type*"
  <xsd:length value="*z*"/>
  </xsd:restriction>
</xsd:simpleType>
 	
where *type* is one of:
  string
  normalizedString
  token
  Name
  NMTOKEN
  language
  NCName
  ID
  IDREF
  ENTITY
  hexBinary
	All mapping levels:
	PIC X(*z*)

<xsd:simpleType>
  <xsd:restriction base="xsd:*type*"
  </xsd:restriction>
</xsd:simpleType>
 	
where *type* is one of:
  duration
  date
  time
  gDay
  gMonth
  gYear
  gMonthDay
  gYearMonth
	All mapping levels:
	PIC X(32)

<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType>	Mapping level 1.2 and below: PIC X(32) Mapping level 2.0 and higher: PIC X(40) Mapping level 3.0 and higher: PIC S9(15) COMP-3 The format is CICS ABSTIME.

<xsd:simpleType>
  <xsd:restriction base="xsd:*type*">
  </xsd:restriction>
</xsd:simpleType>
 	
where *type* is one of:
  byte
  unsignedByte
	All mapping levels:
	PIC X DISPLAY

<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9999 COMP-5 SYNC or PIC S9999 DISPLAY

Schema simple type	COBOL data description
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9999 COMP-5 SYNC or PIC 9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9(18) COMP-3
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9(9) COMP-5 SYNC or PIC S9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9(9) COMP-5 SYNC or PIC 9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC S9(18) COMP-5 SYNC or PIC S9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9(18) COMP-5 SYNC or PIC 9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*m*" <xsd:fractionDigits value="*n*" </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC 9(*p*)V9(*n*) COMP-3 where $p = m - n$.
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	All mapping levels: PIC X DISPLAY The value x'00' implies false, x'01' implies true.
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: PIC X(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: PIC X(255)

Schema simple type	COBOL data description
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1: PIC X(y) where y =4×(ceil(z/3)). ceil(x) is the smallest integer greater than or equal to x. Mapping level 1.2 and higher: PIC X(z) where the length is fixed. PIC X(16) where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType> where the length is not defined.	
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.1 and below: PIC X(32) Mapping level 1.2 and higher: COMP-1
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.1 and below: PIC X(32) Mapping level 1.2 and higher: COMP-2

Some of the schema types shown in the table map to a COBOL format of COMP-5 SYNC or of DISPLAY, depending on the values (if any) that 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 '9's.
- 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 '9's.

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

C and C++ to XML schema mapping

The DFHLS2SC and DFHLS2WS utility programs support mappings between C and C++ data types and XML schema definitions.

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`.

DFHLS2SC and DFHLS2WS map 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 DFHLS2SC or DFHLS2WS. The `_Packed` qualifier is supported for structures. These restrictions 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 characters are ignored if they are present in the header file.
  Storage class specifiers:
  `auto`
  `register`
  `static`
  `extern`
  `mutable`
  Qualifiers
  `const`
  `volatile`
  `_Export` (C++ only)
  Function specifiers
  `inline` (C++ only)
  `virtual` (C++ only)
  Initial values
- The header file must not contain these items:
  Unions
  Class declarations
  Enumeration data types
  Pointer type variables
  Template declarations
  Predefined macros; that is, macros with names that 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 must use a 32-bit compiler to ensure that an `int` maps to 4 bytes.
- The following C++ reserved keywords are not supported:
  `explicit`

```
using
namespace
typename
typeid
```

- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an `xsd:string` and are processed as null-terminated strings.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to BINARY, character arrays are mapped to `xsd:base64Binary` and are processed as binary data.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to COLLAPSE, `<xsd:whiteSpace value="collapse"/>` is generated for strings.

C and C++ data type	Schema simpleType

`char[z]`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:string">`
	`  <xsd:length value="z"/>`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`char[8]`
 	
Supported at mapping level 3.0 and higher when DATETIME=PACKED15
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:dateTime"`
	`  </xsd:restriction>`
	`</xsd:simpleType>`
	
	The format of the time stamp is CICS ABSTIME.

`char`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:byte">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`unsigned char`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:unsignedByte">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`short`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:short">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`unsigned short`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:unsignedShort">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`int`
`long`
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:int">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`unsigned int`
`unsigned long`
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:unsignedInt">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`long long`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:long">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

C and C++ data type	Schema simpleType

`unsigned long long`	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:unsignedLong">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`bool`
 	
(C++ only)
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:boolean">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`float`
 	
Supported at mapping level 1.2 and higher
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:float">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

`double`
 	
Supported at mapping level 1.2 and higher
	`<xsd:simpleType>`
	`  <xsd:restriction base="xsd:double">`
	`  </xsd:restriction>`
	`</xsd:simpletype>`

XML schema to C and C++ mapping

The DFHSC2LS and DFHWS2LS utility programs support mappings between the XML schema definitions that are included in each web service description and C and C++ data types.

The CICS assistants generate 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, it is replaced by a leading '`X`'.

   For example, `_monthlysummary` becomes `Xmonthlysummary`.
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 one or more 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 112.
6. For attributes, the previous rules are applied to the element name. The prefix `attr_` is added to the element name, and it 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.

DFHSC2LS and DFHWS2LS map schema types 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 and an extra character is allocated for the null-terminator.

- 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	C and C++ data type

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anyType">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 2.0 and below:
	Not supported
	
	Mapping level 2.1 and higher:
	
	Supported

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:anySimpletype">`
`  </xsd:restriction>`
`</xsd:simpleType>`
	Mapping level 1.0:
	Not supported
	
	Mapping level 1.1 and higher:
	
	`char[255]`

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:`*type*`">`
`  <xsd:length value="`*z*`"/>`
`  </xsd:restriction>`
`</xsd:simpleType>`
 	
where *type* is one of:
`string`
`normalizedString`
`token`
`Name`
`NMTOKEN`
`language`
`NCName`
`ID`
`IDREF`
`ENTITY`
`hexBinary`
	All mapping levels:
	`char[`*z*`]`

`<xsd:simpleType>`
`  <xsd:restriction base="xsd:`*type*`">`
` </xsd:restriction>`
`</xsd:simpleType>`
 	
where *type* is one of:
`duration`
`date`
`decimal`
`time`
`gDay`
`gMonth`
`gYear`
`gMonthDay`
`gYearMonth`
	All mapping levels:
	`char[32]`

Schema simpleType	C and C++ data type
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.2 and below: char[32] Mapping level 2.0 and higher: char[40] Mapping level 3.0 and higher: char[8] The format of the time stamp is CICS ABSTIME.
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpletype>	All mapping levels: signed char
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpletype>	All mapping levels: char
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>	All mapping levels: short
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>	All mapping levels: unsigned short
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpletype>	All mapping levels: char[33]
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>	All mapping levels: int
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>	All mapping levels: unsigned int
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>	All mapping levels: long long
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>	All mapping levels: unsigned long long
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpletype>	All mapping levels: bool (C++ only) short (C only)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: char[255]

Schema simpleType	C and C++ data type
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	Mapping level 1.0: Not supported Mapping level 1.1 and higher: char[255]
<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> where the length is not defined	Mapping level 1.1 and below: char[y] where $y = 4 \times (\text{ceil}(z/3))$. ceil($x$) is the smallest integer greater than or equal to x. Mapping level 1.2 and higher: 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.
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.1 and below: char[32] Mapping level 1.2 and higher: float(*)
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	Mapping level 1.0 and below: char[32] Mapping level 1.2 and higher: double(*)

PL/I to XML schema mapping

The DFHLS2SC and DFHLS2WS utility programs support mappings between PL/I data structures and XML schema definitions. Because the Enterprise PL/I compiler and older PL/I compilers differ, two language options are supported: PLI-ENTERPRISE and PLI-OTHER.

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.

DFHLS2SC and DFHLS2WS map 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 DFHLS2SC or DFHLS2WS. The following restrictions also apply:

- Data items with the COMPLEX attribute are not supported.
- Data items with the FLOAT attribute are supported at a mapping level of 1.2 or higher. Enterprise PL/I FLOAT IEEE is not supported.

- VARYING and VARYINGZ pure DBCS strings are supported at a mapping level of 1.2 or higher.
- 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.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to NULL, character arrays are mapped to an `xsd:string` and are processed as null-terminated strings; this mapping does not apply for Enterprise PL/I.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to BINARY, character arrays are mapped to `xsd:base64Binary` and are processed as binary data.
- If the **MAPPING-LEVEL** parameter is set to 1.2 or higher and the **CHAR-VARYING** parameter is set to COLLAPSE, `<xsd:whiteSpace value="collapse"/>` is generated for strings.

DFHLS2SC and DFHLS2WS do not fully implement the padding algorithms of PL/I; therefore, you must declare padding bytes explicitly in your data structure. DFHLS2SC and DFHLS2WS issue a message if they detect that padding bytes are missing. Each top-level structure must start on a double-word boundary and each byte in 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 the fields in the following order:

1. FIELD3 is aligned first because it has the strongest boundary requirements.
2. FIELD2 is aligned at the fullword boundary immediately before FIELD3.
3. FIELD1 is aligned 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 that 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 (*n*) where *n* ≤ 7	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
FIXED BINARY (*n*) where 8 ≤ *n* ≤ 15	<xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType>
FIXED BINARY (*n*) where 16 ≤ *n* ≤ 31	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>
FIXED BINARY (*n*) where 32 ≤ *n* ≤ 63 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where *n* ≤ 8 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where 9 ≤ *n* ≤ 16 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where 17 ≤ *n* ≤ 32 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) where 33 ≤ *n* ≤ 64 **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType>
FIXED DECIMAL(*n*,*m*)	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>
FIXED DECIMAL(15) Supported at mapping level 3.0 and higher when DATETIME=PACKED15	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> The format of the time stamp is CICS ABSTIME.
BIT(*n*) where *n* is a multiple of 8. Other values are not supported.	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> where *m* = *n*/8

PL/I data description	Schema
CHARACTER(*n*) VARYING and VARYINGZ are also supported at mapping level 1.2 and higher. **Restriction:** VARYINGZ is supported only by Enterprise PL/I	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>
GRAPHIC(*n*) VARYING and VARYINGZ are also supported at mapping level 1.2 and higher. **Restriction:** VARYINGZ is supported only by Enterprise PL/I	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.0 and 1.1, where *m* = 2**n* <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.2 or higher
WIDECHAR(*n*) **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.0 and 1.1, where *m* = 2**n* <xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*n*"/> </xsd:restriction> </xsd:simpleType> at a mapping level of 1.2 or higher
ORDINAL **Restriction:** Enterprise PL/I only	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
BINARY FLOAT(*n*) where *n* <= 21 Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
BINARY FLOAT(*n*) where 21 < *n* <= 53 Values greater than 53 are not supported. Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>
DECIMAL FLOAT(*n*)where *n* <= 6 Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>

PL/I data description	Schema
DECIMAL FLOAT(*n*)where 6 < *n* <= 16 Values greater than 16 are not supported. Supported at mapping level 1.2 and higher.	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

XML schema to PL/I mapping

The DFHSC2LS and DFHWS2LS utility programs support mappings between XML schema definitions and PL/I data structures. Because the Enterprise PL/I compiler and older PL/I compilers differ, two language options are supported: PLI-ENTERPRISE and PLI-OTHER.

The CICS assistants generate 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 attributes 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 more 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 attributes are specified with different values.

   For more information, see "Variable arrays of elements" on page 112.
5. For attributes, the previous rules are applied to the element name. The prefix attr- is added to the element name and is followed by -value or -exist. If the total length is longer than 28 characters, the element name is truncated. For more information, see "Support for XML attributes" on page 116.

   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 31 characters or less.

DFHSC2LS and DFHWS2LS map schema types to PL/I data types according to the following table. 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 and an extra character is allocated for the null-terminator.
- 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 less than 32 768 bytes and to a container if it is more than 32 768 bytes.

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	Mapping level 2.0 and below: Not supported Mapping level 2.1 and higher: Supported
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and higher:CHAR(255)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <xsd:maxLength value="*z*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> where *type* is one of: string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY	All mapping levels:CHARACTER(*z*)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> where *type* is one of: duration date time gDay gMonth gYear gMonthDay gYearMonth	All mapping levels:CHAR(32)
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.2 and below: CHAR(32) Mapping level 2.0 and higher: CHAR(40) Mapping level 3.0 and higher: FIXED DECIMAL(15) The format of the time stamp is CICS ABSTIME.

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*y*"/> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: BIT(*z*) where *z* = 8 ×*y* and *z* < 4095 bytes. CHAR(*z*) where *z* = 8 ×*y* and *z* > 4095 bytes. Mapping levels 1.2 and higher: CHAR(*y*)
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** SIGNED FIXED BINARY (7) **Other PL/I** FIXED BINARY (7)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** UNSIGNED FIXED BINARY (8) **Other PL/I** FIXED BINARY (8)
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** SIGNED FIXED BINARY (15) **Other PL/I** FIXED BINARY (15)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** UNSIGNED FIXED BINARY (16) **Other PL/I** FIXED BINARY (16)
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** FIXED DECIMAL(31,0) **Other PL/I** FIXED DECIMAL(15,0)
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	All mapping levels: **Enterprise PL/I** SIGNED FIXED BINARY (31) **Other PL/I** FIXED BINARY (31)

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** UNSIGNED FIXED BINARY(32) Mapping level 1.2 and higher: **Enterprise PL/I** CHAR(*y*) where *y* is a fixed length that is less than 16 MB. All mapping levels: **Other PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** SIGNED FIXED BINARY(63) Mapping level 1.2 and higher: **Enterprise PL/I** CHAR(*y*) where *y* is a fixed length that is less than 16 MB. All mapping levels: **Other PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** UNSIGNED FIXED BINARY(64) Mapping level 1.2 and higher: **Enterprise PL/I** CHAR(*y*) where *y* is a fixed length that is less than 16 MB. All mapping levels: **Other PL/I** BIT(64)

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	Mapping level 1.1 and below: **Enterprise PL/I** SIGNED FIXED BINARY (7) **Other PL/I** FIXED BINARY (7) Mapping level 1.2 and higher: **Enterprise PL/I** BIT(7) BIT(1) **Other PL/I** BIT(7) BIT(1) where BIT(7) is provided for alignment and BIT(1) contains the Boolean mapped value.
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>	All mapping levels:FIXED DECIMAL(*n*,*m*)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	All mapping levels:CHAR(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	All mapping levels:CHAR(255)
<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> where the length is not defined	Mapping level 1.0: Not supported Mapping level 1.1: CHAR(*z*) where $z = 4 \times (\text{ceil}(y/3))$. ceil($x$) is the smallest integer greater than or equal to x. Mapping level 1.2 and higher: CHAR(*y*) where the length is fixed. CHAR(16) where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.

Schema	PL/I data description
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	Mapping levels 1.0 and 1.1: CHAR(32) Mapping level 1.2 and higher: **Enterprise PL/I** DECIMAL FLOAT(6) HEXADEC **Other PL/I** DECIMAL FLOAT(6)
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	Mapping levels 1.0 and 1.1: CHAR(32) Mapping level 1.2 and higher: **Enterprise PL/I** DECIMAL FLOAT(16) HEXADEC **Other PL/I** DECIMAL FLOAT(16)

Variable arrays of elements

XML can contain an array with varying numbers of elements. In general, WSDL documents and XML schemas that contain varying numbers of elements do not map efficiently into a single high-level language data structure. CICS uses container-based mappings or inline mappings to handle varying numbers of elements in XML.

An array with a varying number of elements is represented in the XML schema by using the `minOccurs` and `maxOccurs` attributes on the element declaration:

- The `minOccurs` attribute specifies the minimum number of times that the element can occur. It can have a value of 0 or any positive integer.
- The `maxOccurs` attribute specifies the maximum number of times that 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 no upper limit applies to the number of times the element can occur.
- The default value for both attributes is 1.

This example denotes an 8-byte string that is optional; that is, it can occur never or once in the application XML or SOAP message:

```
<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>
```

The following example denotes an 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, WSDL documents 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 transforms XML to application data, it populates these structures with the application data and the application reads them.
- When CICS transforms the application data to XML, it reads the application data in the structures that have been populated by the application.

The format of these data structures is best explained with a series of examples. The XML can be from a SOAP message or from an application. 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
```

Varying number of elements at mapping level 2 and below

This 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 main data structure contains a declaration of two fields. When CICS transforms the XML to binary data, the first field `component-num` contains the number of times that the element appears in the XML, and the second field, `component-cont`, contains the name of a container:

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

A second data structure contains the declaration of the element itself:

```
01 DFHWS-component
  02 component PIC X(8)
```

You must examine the value of `component-num` (which will contain a value in the range 1 to 5) to find out how many times the element occurs. The element contents

are in the container named in component-cont; the container holds an array of elements, where each element is mapped by the DFHWS-component data structure.

If minOccurs="0" and maxOccurs="1", the element is optional. To process the data structure in your application program, you must examine the value of component-num:

- If it is zero, the message has no component element 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.

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.

Varying number of elements at mapping level 2.1 and above

At mapping level 2.1 and above, you can use the **INLINE-MAXOCCURS-LIMIT** parameter in the CICS assistants. The **INLINE-MAXOCCURS-LIMIT** parameter specifies the way that varying numbers of elements are handled. The mapping options for varying numbers of elements are container-based mapping, described in "Varying number of elements at mapping level 2 and below" on page 113, or inline mapping. The *value* of this parameter can be a positive integer in the range 0 - 32767:

- The default value of **INLINE-MAXOCCURS-LIMIT** is 1, which ensures that optional elements are mapped inline.
- A value of 0 for the **INLINE-MAXOCCURS-LIMIT** parameter prevents inline mapping.
- If maxOccurs is less than or equal to the value of **INLINE-MAXOCCURS-LIMIT**, inline mapping is used.
- If maxOccurs is greater than the value of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used.

Mapping varying numbers of elements inline results in the generation of both an array, as happens with the fixed occurrence example above, and a counter. The component-num field indicates how many instances of the element are present, and these are pointed to by the array. For the example shown in "Varying number of elements at mapping level 2 and below" on page 113, when **INLINE-MAXOCCURS-LIMIT** is less than or equal to 5, the generated data structure is like this:

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

The first field, component-num, is identical to the output for the container-based mapping example in the previous section. The second field contains an array of length 5 which is large enough to contain the maximum number of elements that can be generated.

Inline mapping differs from container-based mapping, which stores the number of occurrences of the element and the name of the container where the data is placed, because it stores all the data in the current container. Storing the data in the current container will generally improve performance and make inline mapping preferable.

Nested variable arrays

Complex WSDL documents and XML schemas can contain variably recurring elements, which in turn contain variably recurring elements. In this case, the structure described extends beyond the two levels described in the examples.

This example illustrates an optional element called <component2> that is nested in a mandatory element called <component1>, where the mandatory element 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-5
05 component1-cont PIC X(16)
```

However, the second data structure contains these elements:

```
01 DFHWS-component1
  02 component2-num PIC S9(9) COMP-5
  02 component2-cont PIC X(16)
```

A third-level structure contains these elements:

```
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 structure, consider the fragment of XML that matches the example:

```
<component1><component2>string1</component2></component1>
<component1><component2>string2</component2></component1>
<component1></component1>
```

<component1> occurs three times. 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. The container named in component1-cont has 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 are 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

Optional structures and xsd:choice

DFHWS2LS and DFHSC2LS support the use of maxOccurs and minOccurs on <xsd:sequence>, <xsd:choice>, and <xsd:all> elements only at mapping level 2.1 and above, where the minOccurs and maxOccurs attributes are set to minOccurs="0" and maxOccurs="1".

The assistants generate mappings that treat these elements as though each child element in them is optional. When you implement an application with these elements, ensure that invalid combinations of options are not generated by the application. Each of the elements has its own count field in the generated languages structure, these fields must either all be set to "0" or all be set to"1". Any other combination of values is invalid, except for with <xsd:choice> elements.

<xsd:choice> elements indicate that only one of the options in the element can be used. It is supported at all mapping levels. The assistants handle each of the options in an <xsd:choice> as though it is in an <xsd:sequence> element with minOccurs="0" and maxOccurs="1". Take care when you implement an application using the <xsd:choice> element to ensure that invalid combinations of options are not generated by the application. Each of the elements has its own count field in the generated languages structure, exactly one of which must be set to '1' and the others must all be set to '0'. Any other combination of values is invalid, except when the <xsd:choice> element is itself optional, in which case it is valid for all the fields to be set to '0'.

Support for XML attributes

XML schemas can specify attributes that are allowed or required in XML. The CICS assistant utilities DFHWS2LS and DFHSC2LS ignore XML attributes by default. To process XML attributes that are defined in the XML schema, the value of the **MAPPING-LEVEL** parameter must be 1.1 or higher.

Optional attributes

Attributes can be optional or required and can be associated with any element in a SOAP message or XML for an application. 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 1 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 with which they are associated. 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 maps 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

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 transforms the data to XML, the following runtime processing takes place:

- If the existence flag is set, the attribute is transformed and included in the XML.
- If the existence flag is not set, the attribute is not included in the XML.

Required attributes and runtime processing

For every attribute that is required, only the value field is generated in the appropriate language structure.

If the attribute is present in the XML, the value is mapped. If the attribute is not present, the following processing occurs:

- If the application is a Web service provider, CICS generates a SOAP fault message indicating an error in the client SOAP message.
- If the application is a Web service requester, CICS issues a message and returns a conversion error response with a RESP2 code of 13 to the application.
- If the application is using the **TRANSFORM XMLTODATA** command, CICS issues a message and returns an invalid request response with a RESP2 code of 3 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. When an application uses the **TRANSFORM DATATOXML** command, CICS also transforms the attribute and includes it in the XML.

The nillable attribute

The nillable attribute is a special attribute that can appear on an `xsd:element` in an XML schema. It specifies that the `xsi:nil` attribute is valid for the element in XML. 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, it is mapped as a required attribute that takes a Boolean value.

When CICS receives a SOAP message or has to transform XML for an application that contains an `xsi:nil` attribute, the value of the attribute is true or false. If the value is true, the application must ignore the values of the element or nested elements in the scope of the `xsi:nil` attribute.

When CICS produces a SOAP message or XML based on the contents of a COMMAREA or container for which the value for the `xsi:nil` attribute is true, the following processing occurs:

- The `xsi:nil` attribute is generated into the XML or SOAP message.
- The value of the associated element is ignored.
- Any nested elements within the element are ignored.

SOAP message example

Consider the following example XML schema, which could be part of a WSDL document:

```
<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>
```

Here is an example of a partial SOAP message that conforms to this schema:

```
<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 maps to these elements:

```
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)
```

Support for <xsd:any> and `xsd:anyType`

DFHWS2LS and DFHSC2LS support the use of <xsd:any> and `xsd:anyType` in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. `xsd:anyType` is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

Before you can use <xsd:any> and `xsd:anyType` with the CICS assistants, set the following parameters:

- Set the **`MAPPING-LEVEL`** parameter to `2.1` or higher.
- For a web service provider application, set the **`PGMINT`** parameter to CHANNEL.

<xsd:any> example

This example uses an <xsd:any> element to describe some optional unstructured XML content following the `"Surname"` tag in the `"Customer"` tag:

```
<xsd:element name="Customer">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="FirstName" type="xsd:string"/>
   <xsd:element name="Surname" type="xsd:string"/>
   <xsd:any minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

An example SOAP message that conforms to this XML schema is:

```
<xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Customer xmlns="http://www.example.org/anyExample">
      <Title xmlns="">Mr</Title>
      <FirstName xmlns="">John</FirstName>
      <Surname xmlns="">Smith</Surname>
      <ExtraInformation xmlns="http://www.example.org/ExtraInformation">
        <!-- This 'ExtraInformation' tag is associated with the optional xsd:any from the XML schema.
             It can contain any well formed XML. -->
        <ExampleField1>one</ExampleField1>
        <ExampleField2>two</ExampleField2>
      </ExtraInformation>
    </Customer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If this SOAP message is sent to CICS, CICS populates the `Customer-xml-cont` container with the following XML data:

```
<ExtraInformation xmlns="http://www.example.org/ExtraInformation">
  <!-- This 'ExtraInformation' tag is associated with the optional xsd:any from the XML schema.
       It can contain any well formed XML. -->
  <ExampleField1>one</ExampleField1>
  <ExampleField2>two</ExampleField2>
</ExtraInformation>
```

CICS also populates the `Customer-xmlns-cont` container with the following XML namespace declarations that are in scope; these declarations are separated by a space:

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns="http://www.example.org/anyExample"
```

`xsd:anyType` example

The `xsd:anyType` is the base data type from which all simple and complex data types are derived. It does not restrict the data content. If you do not specify a data type, it defaults to `xsd:anyType`; for example, these two XML fragments are equivalent:

```
<xsd:element name="Name" type="xsd:anyType"/>
```

```
<xsd:element name="Name"/>
```

Generated language structures

The language structures generated for <xsd:any> or `xsd:anyType` take the following form in COBOL and an equivalent form for the other languages:

elementName-xml-cont PIC X(16)
: The name of a container that holds the raw XML. When CICS processes an incoming SOAP message, it places the subset of the SOAP message that the <xsd:any> or `xsd:anyType` defines into this container. The application can

process the XML data only natively. The application must generate the XML, populate this container, and supply the container name.

This container must be populated in text mode. If CICS populates this container, it does so using the same variant of EBCDIC as the web service is defined to use. Characters that do not exist in the target EBCDIC code page are replaced with substitute characters, even if the container is read by the application in UTF-8.

elementName-xmlns-cont PIC X(16)
The name of a container that holds any namespace prefix declarations that are in scope. The contents of this container are similar to those of the DFHWS-XMLNS container, except that it includes all the namespace declarations that are in scope and that are relevant, rather than only the subset from the SOAP Envelope tag.

This container must be populated in text mode. If CICS populates this container, it does so using the same variant of EBCDIC as the web service is defined to use. Characters that do not exist in the target EBCDIC code page are replaced with substitute characters, even if the container is read by the application in UTF-8.

This container is used only when processing SOAP messages sent to CICS. If the application tries to supply a container with namespace declarations when an output SOAP message is generated, the container and its contents are ignored by CICS. CICS requires that the XML supplied by the application is entirely self-contained with respect to namespace declarations.

The name of the XML element that contains the <xsd:any> element is included in the variable names that are generated for the <xsd:any> element. In the <xsd:any> example, the <xsd:any> element is nested inside the <xsd:element name="Customer"> element and the variable names that are generated for the <xsd:any> element are `Customer-xml-cont PIC X(16)` and `Customer-xmlns-cont PIC X(16)`.

For an `xsd:anyType` type, the direct XML element name is used; in the `xsd:anyType` example above, the variable names are `Name-xml-cont PIC X(16)` and `Name-xmlns-cont PIC X(16)`.

Support for <xsd:choice>

An <xsd:choice> element indicates that only one of the options in the element can be used. The CICS assistants provide varying degrees of support for <xsd:choice> elements at the various mapping levels.

Support for <xsd:choice> at mapping level 2.2 and higher

At mapping level 2.2 and higher, DFHWS2LS and DFHSC2LS provide improved support for <xsd:choice> elements. The assistants generate a new container that stores the value associated with the <xsd:choice> element. The assistants generate language structures containing the name of a new container and an extra field:

fieldname-**enum**
The discriminating field to indicate which of the options the <xsd:choice> element will use.

fieldname-**cont**
The name of the container that stores the option to be used. A further language structure is generated to map the value of the option.

The following XML schema fragment includes an <xsd:choice> element:

```
<xsd:element name="choiceExample">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element name="option1" type="xsd:string" />
         <xsd:element name="option2" type="xsd:int" />
         <xsd:element name="option3" type="xsd:short" maxOccurs="2" minOccurs="2" />
      </xsd:choice>
   </xsd:complexType>
</xsd:element>
```

If this XML schema fragment is processed at mapping level 2.2 or higher, the assistant generates the following COBOL language structures:

```
   03 choiceExample.
      06 choiceExample-enum         PIC X DISPLAY.
          88 empty                     VALUE X'00'.
          88 option1                   VALUE X'01'.
          88 option2                   VALUE X'02'.
          88 option3                   VALUE X'03'.
      06 choiceExample-cont         PIC X(16).


01 Example-option1.
   03 option1-length                PIC S9999 COMP-5 SYNC.
   03 option1                       PIC X(255).

01 Example-option2.
   03 option2                       PIC S9(9) COMP-5 SYNC.

01 Example-option3.
   03 option3 OCCURS 2             PIC S9999 COMP-5 SYNC.
```

Limitations for <xsd:choice> at mapping level 2.2 and higher

DFHSC2LS and DFHWS2LS do not support nested <xsd:choice> elements; for example, the following XML is not supported:

```
<xsd:choice>
   <xsd:element name ="name1" type="string"/>
   <xsd:choice>
      <xsd:element name ="name2a" type="string"/>
      <xsd:element name ="name2b" type="string"/>
   </xsd:choice>
</xsd:choice>
```

DFHSC2LS and DFHWS2LS do not support recurring <xsd:choice> elements; for example, the following XML is not supported:

```
<xsd:choice maxOccurs="2">
   <xsd:element name ="name1" type="string"/>
</xsd:choice>
```

DFHSC2LS and DFHWS2LS support a maximum of 255 options in an <xsd:choice> element.

Support for <xsd:choice> at mapping level 2.1 and below

At mapping level 2.1 and below, DFHWS2LS provides limited support for <xsd:choice> elements. DFHWS2LS treats each of the options in an <xsd:choice> element as though it is an <xsd:sequence> element that can occur at most once.

Only one of the options in an <xsd:choice> element can be used, so take care when you implement an application using the <xsd:choice> element that you

generate only valid combinations of options. Each of the elements has its own `count` field in the generated languages structure, exactly one of which must be set to 1 and the others must all be set to 0. Any other combination of values is incorrect, except when the `<xsd:choice>` is itself optional, in which case it is valid for all of the fields to be set to 0.

Related reference:

"Support for <xsd:any> and `xsd:anyType`" on page 118
DFHWS2LS and DFHSC2LS support the use of <xsd:any> and `xsd:anyType` in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. `xsd:anyType` is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

"Support for abstract elements and abstract data types" on page 123
The CICS assistants provide support for abstract elements and abstract data types at mapping level 2.2 and higher. The CICS assistants map abstract elements and abstract data types in a similar way to substitution groups.

"Support for substitution groups" on page 122
You can use a substitution group to define a group of XML elements that are interchangeable. The CICS assistants provide support for substitution groups at mapping level 2.2 and higher.

Support for substitution groups

You can use a substitution group to define a group of XML elements that are interchangeable. The CICS assistants provide support for substitution groups at mapping level 2.2 and higher.

At mapping level 2.2 and higher, DFHSC2LS and DFHWS2LS support substitution groups using similar mappings to those used for `<xsd:choice>` elements. The assistant generates an enumeration field and a new container name in the language structure.

The following XML schema fragment includes an array of two `subGroupParent` elements, each of which can be replaced with `replacementOption1` or `replacementOption2`:

```
<xsd:element name="subGroupExample" >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="subGroupParent" maxOccurs="2" minOccurs="2" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="subGroupParent" type="xsd:anySimpleType" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="subGroupParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="subGroupParent" />
```

Processing this XML fragment with the assistant generates the following COBOL language structures:

```
   03 subGroupExample.
      06 subGroupParent OCCURS2.
         09 subGroupExample-enum PIC X DISPLAY.
            88 empty              VALUE X '00'.
            88 replacementOption1  VALUE X '01'.
            88 replacementOption2  VALUE X '02'.
            88 subGroupParent      VALUE X '03'.
         09 subGroupExample-cont PIC X (16).
```

```
01 Example-replacementOption1.
   03 replacementOption1      PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
   03 replacementOption2      PIC S9999 COMP-5 SYNC.

01 Example-subGroupParent.
   03 subGroupParent-length   PIC S9999 COMP-5 SYNC.
   03 subGroupParent          PIC X(255).
```

For more information about substitution groups, see the *W3C XML Schema Part 1: Structures Second Edition specification*: http://www.w3.org/TR/xmlschema-1/#Elements_Equivalence_Class

Related reference:

"Support for <xsd:any> and xsd:anyType" on page 118
DFHWS2LS and DFHSC2LS support the use of <xsd:any> and xsd:anyType in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. xsd:anyType is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

"Support for <xsd:choice>" on page 120
An <xsd:choice> element indicates that only one of the options in the element can be used. The CICS assistants provide varying degrees of support for <xsd:choice> elements at the various mapping levels.

"Support for abstract elements and abstract data types" on page 123
The CICS assistants provide support for abstract elements and abstract data types at mapping level 2.2 and higher. The CICS assistants map abstract elements and abstract data types in a similar way to substitution groups.

Support for abstract elements and abstract data types

The CICS assistants provide support for abstract elements and abstract data types at mapping level 2.2 and higher. The CICS assistants map abstract elements and abstract data types in a similar way to substitution groups.

Support for abstract elements at mapping level 2.2 and higher

At mapping level 2.2 and above, DFHSC2LS and DFHWS2LS treat abstract elements in almost the same way as substitution groups except that the abstract element is not a valid member of the group. If there are no substitutable elements, the abstract element is treated as an <xsd:any> element and uses the same mappings as an <xsd:any> element at mapping level 2.1.

The following XML schema fragment specifies two options that can be used in place of the abstract element. The abstract element itself is not a valid option:

```
<xsd:element name="abstractElementExample" >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="abstractElementParent" maxOccurs="2" minOccurs="2" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="abstractElementParent" type="xsd:anySimpleType" abstract="true" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="abstractElementParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="abstractElementParent" />
```

Processing this XML fragment with the assistant generates the following COBOL language structures:

```
   03 abstractElementExample.
      06 abstractElementParent OCCURS 2.
         09 abstractElementExample-enum  PIC X DISPLAY.
            88 empty                          VALUE X '00'.
            88 replacementOption1             VALUE X '01'.
            88 replacementOption2             VALUE X '02'.
         09 abstractElementExample-cont  PIC X (16).


01 Example-replacementOption1.
   03 replacementOption1                  PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
   03 replacementOption2                  PIC S9999 COMP-5 SYNC.
```

For more information about abstract elements, see the *W3C XML Schema Part 0: Primer Second Edition specification*: http://www.w3.org/TR/xmlschema-0/#SubsGroups

Support for abstract data types at mapping level 2.2 and higher

At mapping level 2.2 and higher, DFHSC2LS and DFHWS2LS treat abstract data types as substitution groups. The assistant generates an enumeration field and a new container name in the language structure.

The following XML schema fragment specifies two alternatives that can be used in place of the abstract type:

```
<xsd:element name="AbstractDataTypeExample" type="abstractDataType" />

<xsd:complexType name="abstractDataType" abstract="true">
   <xsd:simpleContent>
      <xsd:extension base="xsd:string" />
   </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option1">
   <xsd:simpleContent>
      <xsd:restriction base="abstractDataType">
         <xsd:length value="5" />
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option2">
   <xsd:simpleContent>
      <xsd:restriction base="abstractDataType">
         <xsd:length value="10" />
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
```

Processing this XML fragment with the assistant generates the following COBOL language structures:

```
   03 AbstractDataTypeExamp-enum   PIC X DISPLAY.
     88 empty                         VALUE X'00'.
     88 option1                       VALUE X'01'.
     88 option2                       VALUE X'02'.
   03 AbstractDataTypeExamp-cont   PIC X(16).
```

The language structures are generated into separate copy books. The language structure generated for `option1` is generated into one copybook:

```
   03 option1          PIC X(5).
```

The language structure for `option2` is generated into a different copybook:

```
   03 option2          PIC X(10).
```

For more information about abstract data types, see the *W3C XML Schema Part 0: Primer Second Edition specification*: http://www.w3.org/TR/xmlschema-0/#SubsGroups

Related reference:

"Support for <xsd:any> and `xsd:anyType`" on page 118
DFHWS2LS and DFHSC2LS support the use of <xsd:any> and `xsd:anyType` in the XML schema. You can use the <xsd:any> XML schema element to describe a section of an XML document with undefined content. `xsd:anyType` is the base data type from which all simple and complex data types are derived; it has no restrictions or constraints on the data content.

"Support for `<xsd:choice>`" on page 120
An `<xsd:choice>` element indicates that only one of the options in the element can be used. The CICS assistants provide varying degrees of support for `<xsd:choice>` elements at the various mapping levels.

"Support for substitution groups" on page 122
You can use a substitution group to define a group of XML elements that are interchangeable. The CICS assistants provide support for substitution groups at mapping level 2.2 and higher.

How to handle variably repeating content in COBOL

In COBOL, you cannot process variably repeating content by using pointer arithmetic to address each instance of the data. Other programming languages do not have this limitation. This example shows you how to handle variably repeating content in COBOL for a web service application.

This technique also applies to transforming XML to application data using the **`TRANSFORM`** API commands. The following example WSDL document represents a web service with application data that consists of an 8-character string that recurs a variable number of times:

```
<?xml version="1.0"?>
<definitions name="ExampleWSDL"
             targetNamespace="http://www.example.org/variablyRepeatingData/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://www.example.org/variablyRepeatingData/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <types>
   <xsd:schema targetNamespace="http://www.example.org/variablyRepeatingData/">
     <xsd:element name="applicationData">
       <xsd:complexType>
         <xsd:sequence>
           <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>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
   </xsd:schema>
 </types>
```

```
 <message name="exampleMessage">
   <part element="tns:applicationData" name="messagePart"/>
 </message>

 <portType name="examplePortType">
   <operation name="exampleOperation">
     <input message="tns:exampleMessage"/>
     <output message="tns:exampleMessage"/>
   </operation>
 </portType>

 <binding name="exampleBinding" type="tns:examplePortType">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="exampleOperation">
     <soap:operation soapAction=""/>
     <input><soap:body parts="messagePart" encodingStyle="" use="literal"/></input>
     <output><soap:body parts="messagePart" encodingStyle="" use="literal"/></output>
   </operation>
 </binding>
</definitions>
```

Processing this WSDL document through DFHWS2LS generates the following COBOL language structures:

```
          03 applicationData.

            06 component-num                PIC S9(9) COMP-5 SYNC.
            06 component-cont               PIC X(16).


         01 DFHWS-component.
           03 component                    PIC X(8).
```

Note that the 8-character `component` field is defined in a separate structure called `DFHWS-component`. The main data structure is called `applicationData` and it contains two fields, `component-num` and `component-cont`. The `component-num` field indicates how many instances of the `component` data are present and the `component-cont` field indicates the name of a container that holds the concatenated list of `component` fields.

The following COBOL code demonstrates one way to process the list of variably recurring data. It makes use of a linkage section array to address subsequent instances of the data, each of which is displayed by using the `DISPLAY` statement:

```
IDENTIFICATION DIVISION.
       PROGRAM-ID.    EXVARY.

       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      * working storage variables
       01 APP-DATA-PTR             USAGE IS POINTER.
       01 APP-DATA-LENGTH          PIC S9(8) COMP.
       01 COMPONENT-PTR            USAGE IS POINTER.
       01 COMPONENT-DATA-LENGTH    PIC S9(8) COMP.
       01 COMPONENT-COUNT          PIC S9(8) COMP-4 VALUE 0.
       01 COMPONENT-LENGTH         PIC S9(8) COMP.

       LINKAGE SECTION.

      * a large linkage section array
       01 BIG-ARRAY PIC X(659999).

      * application data structures produced by DFHWS2LS
      * this is normally referenced with a COPY statement
       01 DFHWS2LS-data.
```

```
    03 applicationData.
      06 component-num  PIC S9(9) COMP-5 SYNC.
      06 component-cont PIC X(16).

 01 DFHWS-component.
    03 component        PIC X(8).



 PROCEDURE DIVISION USING DFHEIBLK.
 A-CONTROL SECTION.
 A010-CONTROL.

* Get the DFHWS-DATA container
     EXEC CICS GET CONTAINER('DFHWS-DATA')
               SET(APP-DATA-PTR)
               FLENGTH(APP-DATA-LENGTH)
     END-EXEC
     SET ADDRESS OF DFHWS2LS-data TO APP-DATA-PTR

* Get the recurring component data
     EXEC CICS GET CONTAINER(component-cont)
               SET(COMPONENT-PTR)
               FLENGTH(COMPONENT-DATA-LENGTH)
     END-EXEC

* Point the component structure at the first instance of the data
     SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* Store the length of a single component
     MOVE LENGTH OF DFHWS-component TO COMPONENT-LENGTH

* process each instance of component data in turn
     PERFORM WITH TEST AFTER
           UNTIL COMPONENT-COUNT = component-num

* display the current instance of the data
        DISPLAY 'component value is: ' component

* address the next instance of the component data
        SET ADDRESS OF BIG-ARRAY TO ADDRESS OF DFHWS-component
        SET ADDRESS OF DFHWS-component
            TO ADDRESS OF BIG-ARRAY (COMPONENT-LENGTH + 1:1)
        ADD 1 TO COMPONENT-COUNT

* end the loop
     END-PERFORM.

* Point the component structure back at the first instance of
* of the data, for any further processing we may want to perform
     SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* return to CICS.

     EXEC CICS
          RETURN
     END-EXEC

     GOBACK.
```

The code above provides a generic solution to handling variably repeating content. The array, BIG-ARRAY, moves to the start of each component in turn and does not remain fixed at the start of the data. The component data structure is then moved to point at the first byte of the next component. COMPONENT-PTR can be used to recover the start position of the component data if required.

Here is an example SOAP message that conforms to the WSDL document:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
   <applicationData xmlns="http://www.example.org/variablyRepeatingData/">
     <component xmlns="">VALUE1</component>
     <component xmlns="">VALUE2</component>
     <component xmlns="">VALUE3</component>
   </applicationData>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here is the output produced by the COBOL program when it processes the SOAP message:

```
CPIH 20080115103151 component value is: VALUE1
CPIH 20080115103151 component value is: VALUE2
CPIH 20080115103151 component value is: VALUE3
```

Creating a web service provider by using the web services assistant

You can create a service provider application from a web service description that complies with WSDL 1.1 or WSDL 2.0, or from a high-level language data structure. The CICS web services assistant helps you to deploy your CICS applications in a service provider setting.

About this task

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 interfaces in the web service description and the SOAP messages.

You can expose the web service description associated with your service provider using a URI. This URI has the same path as the URI associated with the WEBSERVICE with the suffix `?wsdl` appended. This enables requesters within your business, or external to it, to discover the WSDL files associated with your service providers.

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 that complies with WSDL 1.1 or WSDL 2.0.

Before you begin

Before you can create a service provider application, the following conditions must be satisfied:

- Your web services description must be in a UNIX file in z/OS and you must create a suitable provider mode pipeline in the CICS region.
- You must define to OMVS the user ID under which DFHWS2LS runs.
- The user ID must have read permission to z/OS UNIX and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **WSDL** parameters.
- You must allocate sufficient storage to the user ID for the ID to run Java. You can use any supported version of Java. By default, DFHWS2LS uses the Java version specified in the **JAVADIR** parameter.

About this task

You can use the web services assistant to create language structures from your WSDL for the service provider application. You can also use a WSDL document that is stored in an IBM webSphere Service Registry and Repository (WSRR) server.

Procedure

1. Use the DFHWS2LS batch program to generate a web service binding file and one or more language data structures. DFHWS2LS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider these options when you enable an existing application for web services:
   a. **Which mechanism will CICS use to pass data to the service provider application program?**
      - You can use channels and pass the data in containers or use a COMMAREA. Channels and containers are recommended. Specify them with the **PGMINT** parameter.

   b. **Which language do you want to generate?**
      - DFHWS2LS can generate COBOL, C/C++, or PL/I language data structures. Specify the language using the **LANG** parameter.

   c. **Which mapping level do you want to use?**
      - The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to use the highest level of mapping available. Specify the mapping level with the **MAPPING-LEVEL** parameter.

   d. **Which URI do you want the web service requester to use?**
      - Specify a relative URI using the **URI** parameter; for example, `URI=/my/test/webservice`. The value is used by CICS when it creates the URIMAP resource.

   e. **Under which transaction and user ID will you run the web service request and response?**
      - You can use an alias transaction to run the application to compose a response to the service requester. The alias transaction is attached under the user ID.
      - Specify it with the **TRANSACTION** and **USERID** parameters. These values are used when creating the URIMAP resource. If you do not want to use a specific transaction, do not use these parameters.

   f. **Where is the WSDL document stored?**
      - If you want to retrieve a WSDL document from a WSRR server, instead of from the local file system, you must specify certain parameters in DFHWS2LS.

- As a minimum, you must specify the **WSRR-SERVER** parameter with the location of the WSRR server and the **WSRR-NAME** parameter with the name of the WSDL document that you want to retrieve from WSRR.
- For information about other parameters that you might want to specify if you are using WSRR, see "DFHWS2LS: WSDL to high-level language conversion" on page 66.

g. **If you intend to retrieve your WSDL document from a WSRR server, do you want to do so using a secure connection?**
   - You can use secure socket layer (SSL) encryption by setting the appropriate parameters to interoperate securely with WSRR. For an example, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

- When you submit DFHWS2LS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The language structures are placed in the partitioned data set that you specified with the **PDSLIB** parameter.

2. Copy the generated web service binding file to the `pickup` directory of the provider mode PIPELINE resource that you want to use for your web service application. You must copy the binding file in binary mode.
3. Optional: Copy the web service description or the archive file containing one or more web service descriptions to the same directory as the web service binding file. The archive file must be a .zip file and the file name must match the WSDL file name. With this copy, you can discover the WSDL.
4. Write a service provider application program to interface with the generated language structures and implement the required business logic.
5. Create the WEBSERVICE resource and two URIMAP resources.
   - The WEBSERVICE resource encapsulates the web service binding file in CICS and is used at run time.
   - The first URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.
   - The second URIMAP resource provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI.
     - This URI has the same path as the URI associated with the WEBSERVICE with the suffix `?wsdl` appended.
     - This URIMAP resource is created so that external requesters can use the URI to discover the WSDL archive file or WSDL document.
     - This URIMAP resource is created only if the web service description or the archive file containing one or more web service descriptions has been copied to the same directory as the web service binding file.
     - If the pickup directory contains a WSDL archive file and a WSDL document, the URI returns only the WSDL in the archive file.
     - This function is only available for web services installed using the pipeline scan operation.

   You can create the resources in the following ways:

   a. Using the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and URIMAP resources.
   b. Defining the resources yourself. If you use the CICS Explorer to define a WEBSERVICE resource in a CICS bundle, you can choose to import a web service binding file and a WSDL document or WSDL archive file and include these in the bundle. You can then generate the URIMAP definitions to support the web service and package these in a bundle. For more help

with using the CICS Explorer to create and edit resources in CICS bundles, seeWorking with bundles in the CICS Explorer product documentation.

Results

If you have any problems submitting DFHWS2LS, or the resources do not install correctly, see "Diagnosing deployment errors" on page 593.

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 begin

Before you create a service provider application, make sure that these preconditions have been completed:

- Your high-level language data structures must meet the following criteria:
  - The data structures must be defined separately from the source program; for example, in a COBOL copybook.
  - If your PL/I or COBOL application program uses different data structures for input and output, the data structures must be defined in two different members in a partitioned data set. If the same structure is used for input and output, the structure must be defined in a single member.

    For C and C++, your data structures can be in the same member in a partitioned data set.
- The data structures you process depend on whether you are using a wrapper program:
  - If you are using a wrapper program, the copybook is the interface to the wrapper program.
  - If you are not using a wrapper program, the copybook is the interface to the business logic.
- The language structures must be available in a partitioned data set and you must create a suitable PIPELINE resource in the CICS region:
  - You must define to OMVS the user ID under which DFHLS2WS runs.
  - The user ID must have read permission to z/OS UNIX 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. You can use any supported version of Java. By default, DFHLS2WS uses the Java version specified in the **JAVADIR** parameter.

Procedure

Follow these steps to create a service provider application from a data structure:

1. If the service provider application interface uses channels and many containers, create a channel description document that describes the interface in XML. You must put the channel description document in a suitable directory on z/OS UNIX. CICS uses this document to construct and deconstruct a SOAP message from the containers on a channel. Alternatively, you can use one container on a channel and not create a channel description document.
   - For more information on how to create a channel description document, see "Creating a channel description document" on page 134.

2. Use the DFHLS2WS batch program to generate a web service binding file and web service description from the language structure. DFHLS2WS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider these options when web service enabling an existing application:

   a. **Which mechanism will CICS use to pass data to the service provider application program?**
      - You can use channels and pass the data in containers or use a COMMAREA. Specify the mechanism using the **PGMINT** parameter. If your application interface uses channels and many containers, specify the **REQUEST-CHANNEL** parameter and optionally the **RESPONSE-CHANNEL**. You can only use these parameters when the mapping level is 3.0 or higher.

   b. **Which level of web service description (WSDL document) do you want to generate?**
      - CICS generates descriptions that comply with either WSDL 1.1 or WSDL 2.0 documents. If you want the service provider application to support requests that comply with both levels of WSDL, specify values for the **WSDL_1.1** and **WSDL_2.0** parameters. Ensure that the file names are different when using more than one WSDL parameter. This specification produces two web service descriptions and a binding file.

   c. **Which version of the SOAP protocol do you want to use?**
      - You can specify the version with the **SOAPVER** parameter. You are recommended to use the ALL value, which gives the flexibility to use either SOAP 1.1 or SOAP 1.2 as the binding for the web service description, although you must install the web service into a pipeline that is configured with the SOAP 1.2 message handler. You can use this parameter only when the **MINIMUM-RUNTIME-LEVEL** is 2.0 or higher.

   d. **Which mapping level do you want to use?**
      - The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to specify the highest level of mapping available in the **MAPPING-LEVEL** parameter.

   e. **Which URI do you want the web service requester to use?**
      - Specify an absolute URI using the **URI** parameter; for example, **URI**=http://www.example.org:80/my/test/webservice. The relative part of this address, /my/test/webservice, is used when creating the URIMAP resource. The full URI is used as the<soap:address> element in the web service description. This usage is true for both HTTP and WebSphere MQ URIs.

   f. **Do you want to publish your WSDL document to an IBM WebSphere Service Registry and Repository (WSRR)?**
      - If you want to publish your WSDL document to a WSRR, you must specify the **WSRR-SERVER** parameter in DFHLS2WS. For more information on the parameters that you can specify when using WSRR, see "DFHLS2WS: high-level language to WSDL conversion" on page 53.

   g. **If you intend to publish your WSDL document on a WSRR server, do you want to do so using a secure connection?**
      - You can use secure socket layer (SSL) encryption by setting the appropriate parameters to interoperate securely with WSRR. For an example, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

- When you submit DFHLS2WS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The generated web service description is placed in the location that you specified with the **WSDL**, **WSDL_1.1**, or **WSDL_2.0** parameter.
- If you have used the WSRR parameters in DFHLS2WS, your WSDL document is published to the WSRR server that you specified.

3. Review the generated web service description and perform any necessary customization. For more information, see "Customizing generated web service description documents" on page 135.
4. Copy the web service binding file to the pickup directory of the provider mode pipeline that you want to use for your web service application. You must copy the web service binding file in binary mode.
5. Optional: Copy the web service description or the archive file containing one or more web service descriptions to the same directory as the web service binding file. The archive file must be a .zip file and the file name must match the WSDL file name. With this copy, you can discover the WSDL.
6. Use the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and two URIMAP resources. The WEBSERVICE resource encapsulates the web service binding file in CICS and is used at run time.
   - The first URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.
   - The second URIMAP resource provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI.
     - This URI has the same path as the URI associated with the WEBSERVICE with the suffix `?wsdl` appended.
     - This URIMAP resource is created so that external requesters can use the URI to discover the WSDL archive file or WSDL document.
     - This URIMAP resource is created only if the web service description or the archive file containing one or more web service descriptions has been copied to the same directory as the web service binding file.
     - If the pickup directory contains a WSDL archive file and a WSDL document, the URI returns only the WSDL in the archive file.
     - This function is only available for web services installed using the pipeline scan operation.

   Alternatively, you can define the resources yourself, although this is not recommended.

Results

When you have successfully created the CICS resources, the creation of your service provider application is complete.

If you have any problems submitting DFHLS2WS, or the resources do not install correctly, see "Diagnosing deployment errors" on page 593.

What to do next

Make the web services description available to anyone who needs to develop a web service requester that will access your service.

Creating a channel description document

Create a channel description document when your service provider application uses a channel interface with many containers.

About this task

Use an XML editor to create the channel description document. The schema for the channel description is called `channel.xsd` and is in the `/usr/lpp/cicsts/cicsts52/schemas/channel` directory (where `/usr/lpp/cicsts/cicsts52` is the default install directory for CICS files).

Procedure

1. Create an XML document with a <channel> element and the CICS channel namespace:

   ```
   <channel name="myChannel" xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
   </channel>
   ```

2. Add a <container> element for every container that the application program interface uses on the channel. You must use name, type and use attributes to describe each container. The following example shows six containers with different attribute values:

   ```
   <container name="cont1" type="char" use="required"/>
   <container name="cont2" type="char" use="optional"/>
   <container name="cont3" type="bit" use="required"/>
   <container name="cont4" type="bit" use="optional"/>
   <container name="cont5" type="bit" use="required">
       <structure location="//HLQ.PDSNAME(MEMBER)"/>
   </container>
   <container name="cont6" type="bit" use="optional">
       <structure location="//HLQ.PDSNAME(MEMBER2)"/>
   </container>
   ```

   The structure element indicates that the content is defined in a language structure located in a partitioned data set member.

3. Save the XML document in z/OS UNIX.

Channel schema

The channel description document must conform to the following schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
   xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel" elementFormDefault="qualified">
   <element name="channel"> 1
      <complexType>
         <sequence>
            <element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0"> 2
               <complexType>
                  <sequence>
                     <element name="structure" minOccurs="0"> 3
                       <complexType>
                           <attribute name="location" type="string" use="required"/>
                           <attribute name="structure" type="string" use="optional"/>
                       </complexType>
                     </element>
                  </sequence>
                  <attribute name="name" type="tns:name16Type" use="required"/>
                  <attribute name="type" type="tns:typeType" use="required"/>
                  <attribute name="use" type="tns:useType" use="required"/>
               </complexType>
             </element>
          </sequence>
          <attribute name="name" type="tns:name16Type" use="optional" />
```

```
      </complexType>
   </element>
  <simpleType name="name16Type">
      <restriction base="string">
         <maxLength value="16"/>
      </restriction>
  </simpleType>
  <simpleType name="typeType">
      <restriction base="string">
        <enumeration value="char"/>
        <enumeration value="bit"/>
      </restriction>
  </simpleType>
  <simpleType name="useType">
     <restriction base="string">
        <enumeration value="required"/>
        <enumeration value="optional"/>
     </restriction>
  </simpleType>
</schema>
```

1. This element represents a CICS channel.
2. This element represents a CICS container within the channel.
3. A structure can only be used with 'bit' mode containers. The 'location' attribute indicates the location of a file that maps the contents of container. The 'structure' attribute may be used in C and C++ to indicate the name of structure.

What to do next

Run DFHLS2WS to create the mappings and WSDL document for the web service provider application. DFHLS2WS puts the mappings for the channel in the WSDL document in the order that the containers are specified in the channel description document.

Customizing generated web service description documents

The web service description (WSDL) documents that are generated by DFHLS2WS contain some automatically generated content that might be appropriate for you to change before publishing. Customizing WSDL documents can result in regenerating the web services binding file and, in some cases, writing a wrapper program.

About this task

Follow these steps to customize generated web service description documents:

Procedure

1. If you want to advertise support for HTTPS or communicate using WebSphere MQ, use the **URI** parameter in DFHLS2WS to set an absolute URI. If you have not used the **URI** parameter, you must change the <wsdl:service> and <wsdl:binding> elements 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, use the **URI** parameter in DFHLS2WS to set an absolute URI. If you have not used the **URI** parameter, add the details to the `soap:address` in 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 TCPIPSERVICE resource.

b. If you are using WebSphere MQ as the transport protocol, replace *myQueue* with the name of the appropriate queue.

You can make these changes without requiring any change to the web services binding file.

If you are changing the port name and namespace without regenerating the WSBind file, the monitoring information might be wrong at runtime level 2.1 onwards.

3. Consider whether the automatically generated names in the WSDL document are appropriate for your purposes. You can rename these values:
   - 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 in the WSDL document.

   These values form part of the programmatic interface to which you code a client program. If the generated names are not sufficiently meaningful, maintenance of your application code might be more difficult over a long period of time. Use the DFHLS2WS **REQUEST-NAMESPACE** and **RESPONSE-NAMESPACE** parameters to change the `targetNamespace` of the XML schemas, and the **WSDL-NAMESPACE** parameter to change the `targetNamespace` of the WSDL document.

   If you change any of these values, you must use DFHWS2LS to regenerate the web services binding file. The language structures that are produced will not be the same as your existing language structures, 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 consider removing any fields that are not helpful to a web service client developer:
   - Fields that are used only for output values can be removed from the schema that maps the input data structures.
   - Filler fields.
   - Automatically generated annotations.

   If you make any of these changes, you must 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 must write a wrapper program to map data from the new representation to the old one. This wrapper program needs to perform an **EXEC CICS LINK** command 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.
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. When you override the mapping rules, ensure 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.

Results

If you want to publish your customized WSDL document to an IBM WebSphere Service Registry and Repository (WSRR) server, you must publish it manually using the WSRR interface. You can find more information about WSRR at the following location: WebSphere Service Registry and Repository.

Example

For an example of a WSDL document, see An example of the generated WSDL document.

Sending a SOAP fault

In a service provider, you can use the CICS API to send a SOAP fault to a web service requester. The fault can be issued by the service provider application or by a header processing program in the pipeline.

Before you begin

To use the API, the service provider application must use channels and containers. If the application uses COMMAREAs, write a wrapper program that does use channels and containers to create the SOAP fault message. You can use the API in a header processing program only if it is invoked directly from a CICS-supplied SOAP message handler.

About this task

You might want to issue a SOAP fault to the web service requester if your application logic cannot satisfy the request, for example, or if there is an underlying problem with the request message. Note that CICS does not consider issuing a SOAP fault as an error, so the normal message response pipeline processing takes place rather than any error processing. If you do want to roll back any transactions, you must use the application program.

Procedure

1. In your program, use the **EXEC CICS SOAPFAULT CREATE** command to send a SOAP fault, see SOAPFAULT CREATE.
2. Add the CLIENT or SERVER option on the command. This option indicates where the problem has occurred, either on the client side or on the server.
   - CLIENT indicates that the problem is with the request message that was received.
   - SERVER indicates that the problem occurs when the request message was processed by CICS. This problem might be in an application program, for example, it might be unable to satisfy the request, or it might be an underlying problem that occurs during the pipeline processing.
3. Add the FAULTSTRING option and its length in the FAULTSTRLEN option to provide a summary of why the fault has been issued by the service provider. The contents of this option are in XML. Any data supplied by the application must be in a format that is suitable for direct inclusion in an XML document. The application might have to specify some characters as XML entities. For

example, if the < character is used anywhere other than the start of an XML tag, the application must change it to <. The following example shows an incorrect FAULTSTRING option:

```
dcl msg_faultString char(*) constant('Error: Value A < Value B');
```

The correct way to specify this FAULTSTRING option is as follows:

```
dcl msg_faultString char(*) constant('Error: Value A &lt; Value B');
```

Tip: To avoid using XML entities, you can wrapper the data in an XML CDATA construct. XML processors do not parse character data in this construct. Using this method, you could specify the following FAULTSTRING option:

```
dcl msg_faultString char(*) constant('<![CDATA[Error: Value A < Value B]]>');
```

4. Code the DETAIL option and its length in the DETAILLENGTH option to provide the details of why the fault has been issued by the service provider. The contents of this option are in XML. The same guidance applies to the DETAIL option as to the FAULSTRING option.
5. Optional: If you are invoking the API from a header processing program, define the program in the pipeline configuration file. The header processing program is defined in either the <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> element.

Results

When your program issues this command, CICS creates the SOAP fault response message at the appropriate SOAP level. If your service provider application issues the command, it does not need to create a SOAP response and put it in the DFHRESPONSE container. The pipeline processes the SOAP fault through the message handlers and sends the response to the web service provider.

Example

The **SOAPFAULT CREATE** command has a number of options to provide you with flexibility to respond appropriately to a web service requester. Here is a simple example of a completed command that creates a SOAP fault that can be used for both SOAP 1.1 and SOAP 1.2:

```
EXEC CICS SOAPFAULT CREATE CLIENT
     DETAIL(msg_detail)
     DETAILLENGTH(length(msg_detail))
     FAULTSTRING(msg_faultString)
     FAULTSTRLEN(length(msg_faultString));
```

You can code *msg_detail* and *msg_faultString* with the following values:

```
dcl msg_detail char(*) constant('<ati:ExampleFault xmlns="http://www.example.org/faults"
xmlns:ati="http://www.example.org/faults">Detailed error message goes here.</ati:ExampleFault>');
dcl msg_faultString char(*) constant('Something went wrong');
```

Creating a web service requester using the web services assistant

You can create a service requester application from a web service description that complies with WSDL 1.1 or WSDL 2.0. The CICS web services assistant helps you to deploy your CICS applications in a service requester setting.

Before you begin

Your web services description must be in a file in z/OS UNIX or it must be published on an IBM WebSphere Services Registry and Repository (WSRR) server, and a requester mode pipeline must be installed in the CICS region.

You must allocate sufficient storage to the user ID so that the ID can run Java. You can use any supported version of Java. By default, DFHWS2LS uses the Java version specified in the **JAVADIR** parameter.

About this task

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.

Procedure

1. Use the DFHWS2LS batch program to generate a web service binding file and one or more language structures. Consider these options when creating a service requester application from a web service description:
   - Which mapping level do you want to use? The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to use the highest level of mapping available.
   - Which code page do you want to use when transforming data at run time? If you want to use a specific code page for your application that is different from the code page for the CICS region, use the **CCSID** parameter.
   - Do you want to support a subset of the operations that are declared in the web service description? If you have a very large web service description, and want your service requester application to support only a certain number of operations, use the **OPERATION** parameter to list the ones you want. Each operation must be separated with a space and is case sensitive.
   - Where is the WSDL document stored? If the WSDL document that you want to use as input to DFHWS2LS is stored on a WSRR server, you can retrieve it by running DFHWS2LS with certain parameters specified. Use the **WSRR-SERVER** parameter to specify the location of the WSRR server and use the **WSRR-NAME** parameter to specify the name of the WSDL document that you want to retrieve. For information about other parameters on DFHWS2LS that you might want to use to interact with WSRR, see "DFHWS2LS: WSDL to high-level language conversion" on page 66.
   - If you want to retrieve the WSDL document from a WSRR server, do you want to do so using a secure connection? You can use secure socket layer (SSL) encryption with the web services assistant to interoperate securely with WSRR. For an example, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

   Do not specify parameters such as **PROGRAM**, **URI**, **TRANSACTION**, and **USERID** when you use DFHWS2LS. These parameters apply only to a service provider application and, if specified, cause a provider mode web service binding file to be produced.In addition to the web service binding file, the program generates a language data structure.
2. Check the log file to see whether any problems occurred when DFHWS2LS generated the binding file and language structures. CICS might not support some elements or options in the web service description. If any warning or

error messages are issued, read the advice that is provided and take appropriate action. You might need to rerun the batch program.

3. Copy the web service binding file to the pickup directory of the requester mode pipeline that you want to use for your web service application.
4. Ensure that the PIPELINE resource is configured for service requester applications. The value of the **MODE** parameter shows whether the pipeline supports requester or provider web service applications.
5. Ensure that the correct SOAP protocol is supported in the pipeline for your web service. The **SOAPLEVEL** parameter indicates which version is supported. In service requester mode, the binding of the web service must match the version of SOAP that is supported in the pipeline. You cannot install a web service with a SOAP 1.1 binding into a service requester pipeline that supports SOAP 1.2.
6. Ensure that the configured timeout for the pipeline is suitable for your service requester application. The timeout is displayed as the value of the RESPWAIT attribute on the PIPELINE resource. If no timeout is specified on the pipeline, the default for the transport is used.
   - The default timeout for HTTP is 10 seconds.
   - The default timeout for WebSphere MQ is 60 seconds.

   Each transaction in the CICS region has a dispatcher timeout. If this value is less than the default for either protocol, the timeout occurs with the dispatcher.
7. Optional: Copy the web service description to the same pickup directory as the web service binding file, so that you can turn on validation for the web service at run time.
8. Create the WEBSERVICE resource. This resource encapsulates the web service binding file in CICS and is used at run time.

   You can do this in the following ways:

   a. Using the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource.
   b. Defining the resource yourself. If you use the CICS Explorer to define a WEBSERVICE resource in a CICS bundle, you can choose to import a web service binding file and a WSDL document or WSDL archive file and include these in the bundle. You can then generate URIMAP definitions to support the web service and package these in a bundle. For more help with using the CICS Explorer to create and edit resources in CICS bundles, see Working with bundles in the CICS Explorer product documentation.
9. Write a wrapper program that you can substitute for your communications logic. Use the language data structure generated in step 1 to write your wrapper program. Use an **EXEC CICS INVOKE SERVICE** command in your wrapper program to communicate with the web service. The command includes these options:
   - The URI of the web service
   - The operation for which the web service is being called

   When you call the web service, you can specify a URIMAP resource that contains the information about the URI of the web service. You can specify this information directly on the INVOKE SERVICE command instead of using a URIMAP resource. However, using a URIMAP resource means that you do not need to recompile your applications if the URI of a service provider changes. With a URIMAP resource you can also choose to implement connection pooling, where CICS keeps the client connection open after use, so that it can be reused by the application for subsequent requests, or by another application

that calls the same service. The PIPELINE SCAN command does not create URIMAP resources for a service requester, so you must define the URIMAP resource yourself following the instructions in Creating a URIMAP resource for CICS as a HTTP client in Developing applications.

Results

When you have successfully created the CICS resources, the creation of your service requester application is complete.

Checking the configuration of a PIPELINE resource

You can check the configuration of a PIPELINE with the following interfaces:

CICS Explorer

The CICS Explorer administration views

Use the Pipelines view.

CICSPlex SM

The PIPELINE definitions view

CEMT

The INQUIRE PIPELINE command

The CICS SPI

The INQUIRE PIPELINE command

Creating a web service using tooling

Instead of using the web services assistant JCL, you can use Rational Developer for z Systems or write your own Java program to create the required files in CICS.

Procedure

1. You have two choices:
   - Use the Rational Developer for z Systems tool to create a web service binding file and the web service description or language structures. For more information about this tool, see Rational Developer for System z.
   - Write your own Java program, using the provided API, to invoke the web services assistant. This API is described in the Web services assistant: Class Reference Javadoc. It includes comments that explain the classes, and sample code is provided to give an example of how you might invoke the web services assistant. The Javadoc also contains a complete list of the JAR files that are required and their location in z/OS UNIX.

     You can run your Java program on the z/OS, Windows, or Linux platform. If you run the program on Windows or Linux, transfer the generated web services binding file to a suitable pickup directory in binary mode using FTP or an equivalent process.
2. Optional: If you are generating a web service description from a language structure, review the file and perform any necessary customization. For more information, see “Customizing generated web service description documents” on page 135.

3. Deploy the generated web service binding file into a suitable pipeline pickup directory.
4. Optional: Copy the web service description into the pickup directory of the pipeline, so that you can perform validation of the web service to check that it is working as expected.
5. If you started with a web service description, write a service provider or requester application program to interface with the generated language structures.
6. Run a **PIPELINE SCAN** command to automatically create the required CICS resources.

Creating your own XML-aware web service applications

If you decide not to use the CICS-supplied data mappings, you can write your own XML-aware data applications in two ways instead. You can either use the **XML-ONLY** parameter on DFHWS2LS or you can write your own application without using any of the tooling. Using the **XML-ONLY** parameter is the most straightforward way to configure the CICS pipeline process to pass the XML data to the application to be handled.

About this task

Writing your own XML-aware applications involves writing code to both parse and generate XML documents. One way to write your own XML-aware application uses the `XML PARSE` and `XML GENERATE` statements in COBOL. Another way to write your own XML-aware applications uses other IBM tools; for example, you can use the Rational Developer for z Systems for tool to generate COBOL XML converter programs that can be invoked from your applications.

Creating an XML-aware service provider application

Your XML-aware service provider application must work with the containers that are passed to it and handle the data conversion between the XML and the program language.

About this task

The following steps guide you through the creation of your XML-aware application, including the decision about the use of any of the CICS tooling.

Procedure

1. Decide if you want to generate a web service binding file for your XML-aware application using DFHWS2LS. The advantage of generating a web service binding file is that you can use CICS services, such as validation, to test your web service and CICS monitoring using global user exits.
   - If you want to generate a web service binding file, run DFHWS2LS specifying the **XML-ONLY** parameter and a **MINIMUM-RUNTIME-LEVEL** of 2.1 or higher. The web service binding file enables the application program to work directly with the contents of the DFHWS-BODY container. In all other respects, the generated binding file shares the same deployment characteristics and the same runtime behavior as a file generated without the **XML-ONLY** parameter, including parsing of the XML during SOAP message handling. To prevent this parsing, you must not specify SOAP message handlers in your pipeline configuration file.

- If you do not want to use a web service binding file, configure your service provider pipeline so that the web service request reaches your XML-aware application. You can either configure the terminal handler in the pipeline configuration file to use your XML-aware application program or create a message handler that dynamically switches to your application depending on the URI that is received in the pipeline.

2. Write your application to handle the web service request that is held in the following containers:

   DFHWS-BODY
   The contents of the SOAP body for an inbound SOAP request when the pipeline includes a CICS-provided SOAP message handler.

   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.

   When you code API commands to work with the containers, do not 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.

3. Optional: Your application can also use additional containers that are available to message handlers in the pipeline, as well as any other containers that the message handlers create as part of their processing. For a complete list of containers, see "Containers used in the pipeline" on page 290.

4. When your application has processed the request, construct a web service response using the following containers:

   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 instead of using the CICS-provided SOAP message handler.

   If you supply a SOAP body in container DFHWS-BODY, DFHRESPONSE is ignored.

   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 constructs the full SOAP message containing the body.

   Your program must create this container, even if the request and response are identical. If you do not, CICS issues an internal server error.

   You can also use any of the other containers to pass information that your pipeline needs for processing the outbound response.

If 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, because the message handler checks only whether the container is present or not.

5. Create a URIMAP resource. If you are using the **XML-ONLY** parameter and you have specified a value for the **URI** parameter of DFHWS2LS, the URIMAP is created automatically for you during the PIPELINE SCAN process.

Creating an XML-aware service requester application

Your XML-aware web service requester application handles the data conversion between the XML and the programming language and populates the control containers in the pipeline.

Before you begin

You can write your own XML-aware service requester application using the **XML-ONLY** parameter on DFHWS2LS or you can write it without using any of the tooling. The most straightforward way to write your own XML-aware service requester application is by using the **XML-ONLY** parameter on DFHWS2LS; the **XML-ONLY** parameter is available at runtime level 2.1 and later.

About this task

Using the **XML-ONLY** parameter results in the generation of a WSBind file that instructs CICS that the application will work directly with the contents of the DFHWS-BODY container. The generated WSBind file must be installed into a requester mode PIPELINE to create a requester mode WEBSERVICE resource. The application must generate XML for the body of the web service request and store it in the DFHWS-BODY container. It must then call the **EXEC CICS INVOKE SERVICE** command. The outbound message is sent to the web services provider. The body of the response message is also in the DFHWS-BODY container after the call completes.

The XML of the response messages is parsed during SOAP message handling. To prevent this parsing, you must not specify SOAP message handlers in your pipeline configuration file.

XML-aware requester applications can receive SOAP Fault messages back from the remote provider mode application. In this case, the requester application is responsible for interpreting the SOAP Fault and distinguishing it from a regular response message. If the **INVOKE SERVICE** command is used with an **XML-ONLY** WEBSERVICE, CICS does not set the response code which is normally used to indicate that a SOAP Fault was received.

If you are writing your own XML-aware service requester application without using the **XML-ONLY** option, complete the following steps:

Procedure

1. Create a channel and populate it with containers. The control containers must all be populated in CHAR mode. Provide the following information in each container:

   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 constructs 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. The pipeline must not include a CICS-provided SOAP message handler to avoid duplicate SOAP headers being sent in the outbound message.

If you supply a SOAP body in container DFHWS-BODY, DFHREQUEST must 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.

Tip: If you add container DFHWS-NOABEND to the channel, when DFHPIRT is called any abends will not be issued from within DFHPIRT. This is useful if you are running a C/C++ program because you can handle errors via the DFHERROR container.

2. Link to program DFHPIRT. Use this command:

```
EXEC CICS LINK PROGRAM(DFHPIRT) CHANNEL(userchannel)
```

where *userchannel* is the channel that holds your containers. The outbound message is processed by the message handlers and header processing programs in the pipeline and sent to the web service provider.

3. Retrieve the containers that contain the web service response from the same channel. The response from the web service provider might be a successful response or a SOAP fault. The web service requester application must be able to handle both types of response from the service provider. The complete response is contained in the following containers:

DFHRESPONSE
The complete response, including the envelope for a SOAP response, received from the web service provider.

DFHWS-BODY
When the pipeline includes a CICS-provided SOAP message handler, the contents of the SOAP body.

DFHERROR
Error information from the pipeline.

Note: In some error cases DFHWS-BODY might not be updated. You must check DFHRESPONSE for a SOAP fault.

Using Java with web services

You can use Java to create web service applications. Different techniques are used to create these applications compared with the techniques used with other programming languages.

For most non-Java programming languages, you use the web services assistants to enable applications. Using the web services assistant means CICS will convert the data from the web service into a form suitable for the application and place it into a container or COMMAREA. You can use the web services assistant with Java applications, however, the following tasks provide more suitable methods for creating Java web services for Java applications.

Deploying a Java provider-mode web service in an Axis2 JVM server

You can deploy an Axis2 application as a provider mode web service in CICS. These applications are typically generated using JAX-WS and can be hosted in a Java enabled pipeline.

You might want to deploy Java applications using this method for one of the following reasons:

- You have existing investment using Axis2 Handler interfaces.
- You want to use a CICS pipeline configuration.

Note: Axis2-style applications do not use the WEBSERVICE resources. They interact with CICS using the Axis2 programming model and therefore cannot use the some of the CICS web services support. The following services are not fully supported for Axis2-style applications:

- SOAPFAULT CREATE
- WSACONTEXT GET
- "DFHWS-OPERATION container" on page 302
- "DFHWS-MEP container" on page 302
- "DFHWS-USERID container" on page 308
- "DFHWS-TRANID container" on page 303
- Web services security

Before you begin

You must have a Java application that is suitable for deployment in Axis2, for example a POJO application using JAX-WS. For this task, the following POJO application is used as an example:

```
/**
 * Simple example
 */
@javax.jws.WebService(targetNamespace = "com.ibm.cics.example", name = "pojoExample")
public class TestAxis2
{
    public String getMessage(String input)
    {
        return "CICS got this: '" + input + "'";
    }
}
```

This application specifies the XML namespace that is used to generate the WSDL, and a name to associate with the web service.

The Java code for this application must be compiled, and the JAX-WS generator run, to package the application into a jar file called `TestAxis2.jar`. You can do this by issuing the following code:

```
javac TestAxis2.java
wsgen -cp . TestAxis2 -wsdl
jar -cvf TestAxis2.jar *
```

The JAX-WS generator also creates a WSDL document and the bindings used by Axis2.

About this task

To deploy an Axis2 web service you must create the pipeline infrastructure for your web services. When you have created the pipeline, you can create your web services. You can reuse the created pipeline for as many web services as you need. The following steps describe how to create the pipeline and web services.

Note: No WEBSERVICE resource is created or installed as part of this task.

Procedure

1. Create the pipeline infrastructure.
   a. Create a web service infrastructure for a Java pipeline. For more information, see "Creating the CICS infrastructure for a SOAP service provider" on page 217.
   b. Create an Axis2 repository. To do this, create a copy of the supplied repository located in `$CICS_HOME/lib/pipeline/repository`.
   c. Add the `<repository>` element to your pipeline configuration file. This element must specify the name of the Axis2 repository that you created.
   d. Create and enable a PIPELINE resource.
2. For each web service associated with the pipeline, repeat the following steps to create the web service.
   a. Deploy the Axis2 application to the Axis2 repository. For example, the jar file created in the example must be deployed to a directory called `servicejars` in the repository directory. You must create this directory if it does not exist.
   b. Define and install a URIMAP resource for the web service. The URIMAP resource must specify the URI and PIPELINE resource associated with the web service. The URI must follow the Axis2 naming conventions for URIs. The default Axis2 naming convention is: */name_of_service*`Service.`*name_of_port*`Port/`*suffix*, where *name_of_service* is the name of the web service in the WSDL, *name_of_port* is the name of the port in the WSDL and *suffix* is an optional suffix that you can define. For the preceding example, the following URIMAP resource could be used:

```
Urimap         : EXAMPLE
Group          : EXAMPLE
STatus         : Enabled
USAge          : Pipeline
SCheme         : HTTP
POrt           : No
HOST           : *
PAth           : /TestAxis2Service.pojoExamplePort/example/TestAxis2
TRansaction    : CPIH
PIpeline       : EXAMPLE
```

This example assumes that the PIPELINE resource used is called EXAMPLE.

What to do next

Test that your web services run correctly.

Creating a Java web service that generates and parses XML

You can create Java applications that parse and generate XML themselves. These applications are consistent with XML-aware applications written in other programming languages, but they benefit from using standard Java technologies for processing the XML.

Procedure

1. Create an XML-ONLY WEBSERVICE resource. For more information, see "Creating an XML-aware service requester application" on page 144 or "Creating an XML-aware service provider application" on page 142.
2. Write a Java web service that can parse and generate XML for the body of the SOAP message. You can use various tools, such as the Java 6 Java Architecture for XML Binding (JAXB) library, to help you create a Java web service with these capabilities.
3. Optional: If you are using a provider pipeline and you want to add the capability for a SOAP Fault message to be returned to the requester, use the JCICS SoapFault class to issue the **EXEC CICS SOAPFAULT CREATE** command.
4. Optional: If you are using a requester pipeline, use the JCICS Service class to issue the **EXEC CICS INVOKE SERVICE** command.

Creating a Java web service that has a COBOL interface

You can create Java applications that interact with CICS using the same techniques used in other programming languages. To create these applications, you must write or generate Java code to create structured COMMERA- or container-style data.

Procedure

1. Use DFHWS2LS to create COBOL language structures for the web service.
2. Write a Java web service that generates and parses COBOL language structures. For more information about tools that allow Java programs to access existing CICS application data and links to examples of how to create a Java web service that can generate and parse COBOL language structures, see Interacting with structured data from Java in Developing applications.
3. Optional: If you are using a provider pipeline and you want to add the capability for a SOAP Fault message to be returned to the requester, use the JCICS SoapFault class to issue the **EXEC CICS SOAPFAULT CREATE** command.
4. Optional: If you are using a requester pipeline, use the JCICS Service class to interface with the CICS SERVICE API and issue the **EXEC CICS INVOKE SERVICE** command.

Deploying a requester-mode JAX-WS web service

You can deploy a JAX-WS application as a requester mode web service in CICS. However, these applications do not use the **EXEC CICS INVOKE** command, instead they interact with the remote web services using JAX-WS.

Before you begin

You must have a JVM server configured to support OSGi. For more information, see Setting up a JVM server in Configuring.

About this task

The advantage of deploying a JAX-WS application as a requester mode web service is that you create a platform-independent web service requester application, which uses the zEnterprise Application Assist Processor (zAAP). Using zAAP can reduce the cost of transactions; for more information, see the IBM Redbooks publication: zSeries Application Assist Processor (zAAP) Implementation.

Procedure

1. Create a web service requester application in Java and use an appropriate API, such as the Java API for XML web services (JAX-WS), to call the remote web service.
2. Optional: If you use JAX-WS to start a remote web service, you must also use JAX-WS to generate the SOAP messages, handle the network communication, and process the SOAP response.
3. Deploy your Java application and install it in the JVM server.

What to do next

Test that your web services start correctly.

Deploying a Java provider-mode web service in a Liberty JVM server

You can deploy a web application as a provider mode web service in a Liberty JVM server. These applications are created using Java standards JAX-WS and JAXB.

About this task

CICS TS V5.3 includes the latest WebSphere Application Server Liberty Profile (WLP) V8.5.5 that provides features for the Java API for XML Web Services (JAX-WS) and the Java Architecture for XML Binding (JAXB). Together these technologies enable you to write SOAP web services in Java as part of a CICS application. The following article will show you how to set up Eclipse, test a sample web services project, deploy the sample into CICS, modify the sample to use JCICS and test it with the Web Service Explorer. For details see DeveloperWorks, JAX-WS and JAXB support in CICS TS V5.2 open beta Liberty profile.

You might want to deploy Java applications using this method for one of the following reasons:

- You want to create web services in Java.
- You have complicated WSDL documents that would be difficult to handle using the CICS web services assistants.
- You want to offload the handling of the web service application to the zEnterprise Application Assist Processor (zAAP).

Note: Web applications deployed to a Liberty JVM server do not use the WEBSERVICE or TCPIPSERVICE resources. They interact with web requests using the Liberty HTTP listener and therefore cannot use the facilities of the CICS web services support.

Validating SOAP messages

When you use the CICS web services you can specify that the SOAP messages are to be validated to ensure that they conform to the schema that is contained in the web service description. You can validate both provider and requester mode applications.

Before you begin

During development and testing of your web service deployment, full validation assists in detecting problems in the message exchange between a service requester and a service provider. However, complete validation of the SOAP messages carries a substantial overhead, and it is inadvisable to validate messages in a fully tested production application.

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

About this task

The SOAP message is validated 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 and is validated again against the transformation requirements of CICS. You can use the WSDL file specified in the **WSDLFILE** attribute of the WEBSERVICE resource or a WSDL file contained in the `.zip` file specified in the **ARCHIVEFILE** attribute of the WEBSERVICE resource. If both attributes are specified, the WSDL file in the archive file specified in the **ARCHIVEFILE** attribute is used.

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. Therefore a SOAP message might be successfully validated but then fail in the runtime environment and vice versa.

Procedure

1. Optional: Set up a JVM server in the CICS region. You can run SOAP validation in an OSGi framework or Axis2, but not in a Liberty profile. CICS provides samples to quickly set up a JVM server that uses an OSGi framework. If you are using a Java pipeline, there is no need to define an extra JVM server.
   - When you have a JVM server already defined that you want to use for validation, modify the DFHPIVAL program definition in group DFHPIVAL to reference the name of the JVMSERVER resource. The DFHPIVAL definition is not locked and can be edited. By default, the definition references DFHJVMS.

     or
   - When you want to use the CICS supplied sample you must install the sample JVM server DFHJVMS in group DFH$OSGI. For more information, see Setting up a JVM server in Configuring.
2. Ensure that you have a web service description associated with your WEBSERVICE resource. This association is created for WEBSERVICE resources

that are automatically created when a WSDL file or a `.zip` file that contains one or more WSDL files is present in the pickup directory of the pipeline during a pipeline scan.

For WEBSERVICE definitions that are created with RDO, the web service description is specified with the WSDLFILE attribute.

3. Turn web service validation on by specifying **`VALIDATION=YES`** attribute of the WEBSERVICE resource. You can specify whether validation is required when you define the resource, and you can change this setting after the resource is installed.

Results

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

What to do next

Turn validation off when you no longer need it.

Changing the validation status of a web service

You can change the validation status of a web service with the following interfaces:

CICS Explorer

The CICS Explorer administration views

Use the **`Validation Status`** attribute in the Web Services view.

CICSPlex SM

The WEBSERVICE definitions view

CEMT

The SET WEBSERVICE command

The CICS SPI

The SET WEBSERVICE command

Chapter 8. Creating a JSON web service

You can expose existing CICS applications as JSON web services and create new CICS applications to act as JSON web service providers.

Before you begin

Before you begin to create a JSON web service, you must configure your CICS system to support JSON web services. For more information, see "Creating the CICS infrastructure for a JSON service provider" on page 221.

About this task

The CICS JSON assistant is a supplied utility that helps you to create the necessary artifacts for a new JSON web service provider application, or to enable an existing application as a JSON web service provider.

The CICS JSON assistant can create a JSON schema from a high-level language structure or a high-level language structure from an existing JSON schema; it supports COBOL, C/C++, and PL/I. It also generates information that is used to enable automatic runtime conversion of the JSON messages to containers and COMMAREAs, and vice versa. This information is used by the CICS JSON web services support during pipeline processing.

Create your JSON web service, as described in the following procedure, and validate that it works correctly:

Procedure

1. Create a JSON web service. Use the JSON assistant to create the JSON schema or language structures and deploy them into CICS. Use the **PIPELINE SCAN** command to automatically create the required CICS resources.
2. Start the JSON web service to test that it works as you intended.

What to do next

These steps are explained in more detail in the following topics.

The CICS JSON assistant

The CICS JSON assistant is a set of batch utilities that creates a mapping between JSON schema and language structures. This mapping is used by CICS at runtime to do the transformation between JSON and application data. 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 JSON 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 JSON message and the application program's data structure.

The assistant can create a JSON schema from a high-level language structure or a high-level language structure from an existing JSON schema, and supports

COBOL, C/C++, and PL/I. It also generates information used to enable automatic runtime conversion of the JSON messages to containers and COMMAREAs, and vice versa.

The CICS JSON assistant comprises two utility programs:

DFHLS2JS
: Generates a web service binding file from a language structure. This utility also generates a JSON schema.

DFHJS2LS
: Generates a web service binding file from a JSON schema. 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.

For more information about the JSON assistant utility programs and data mappings, see the following topics.

DFHLS2JS: High-level language to JSON schema conversion for request-response services

The DFHLS2JS procedure generates a JSON schema file from a high-level language data structure. You can use DFHLS2JS when you expose a CICS application program as a service provider.

The job control statements for DFHLS2JS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

The DFHLS2JS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

The relevant usage mode for the DFHLS2JS or DFHJS2LS procedure depends on your requirements:

- DFHLS2JS: High-level language to JSON schema conversion for linkable interface
- DFHJS2LS: JSON schema to high-level language conversion for linkable interface
- DFHLS2JS: High-level language to JSON schema conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for RESTful services

Job control statements for DFHLS2JS

JOB Starts the job.

EXEC Specifies the procedure name (DFHLS2JS).

INPUT.SYSUT1 DD
: Specifies the input. The input parameters are typically 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 DFHLS2JS:

JAVADIR=*path*
Specifies the name of the Java directory that is used by DFHLS2JS. The value of this parameter is appended to `/usr/lpp/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 a prefix that extends the z/OS UNIX directory path that is used on other parameters, or `''` (empty string) if no prefix is used.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHPREF** parameter.

SERVICE=*value*
Use this parameter only when directed to do so by IBM Support.

TMPDIR=*tmpdir*
Specifies the location of a directory in z/OS UNIX that DFHLS2JS uses as a temporary workspace. The user ID used to run the job must have read and write permission to this directory.

The default value is `/tmp`.

TMPFILE=*tmpprefix*
Specifies a prefix that DFHLS2JS uses to construct the names of the temporary workspace files.

The default value is `LS2JS`.

USSDIR=*path*
Specifies the name of the CICS TS directory in the UNIX System Services file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*. This must be specified as `'.'` (period) if the default is used.

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

The temporary workspace

DFHLS2JS creates the following three temporary files at run time:

tmpdir`/`*tmpprefix*`.in`

tmpdir`/`*tmpprefix*`.out`

tmpdir`/`*tmpprefix*`.err`

Where:

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

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

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

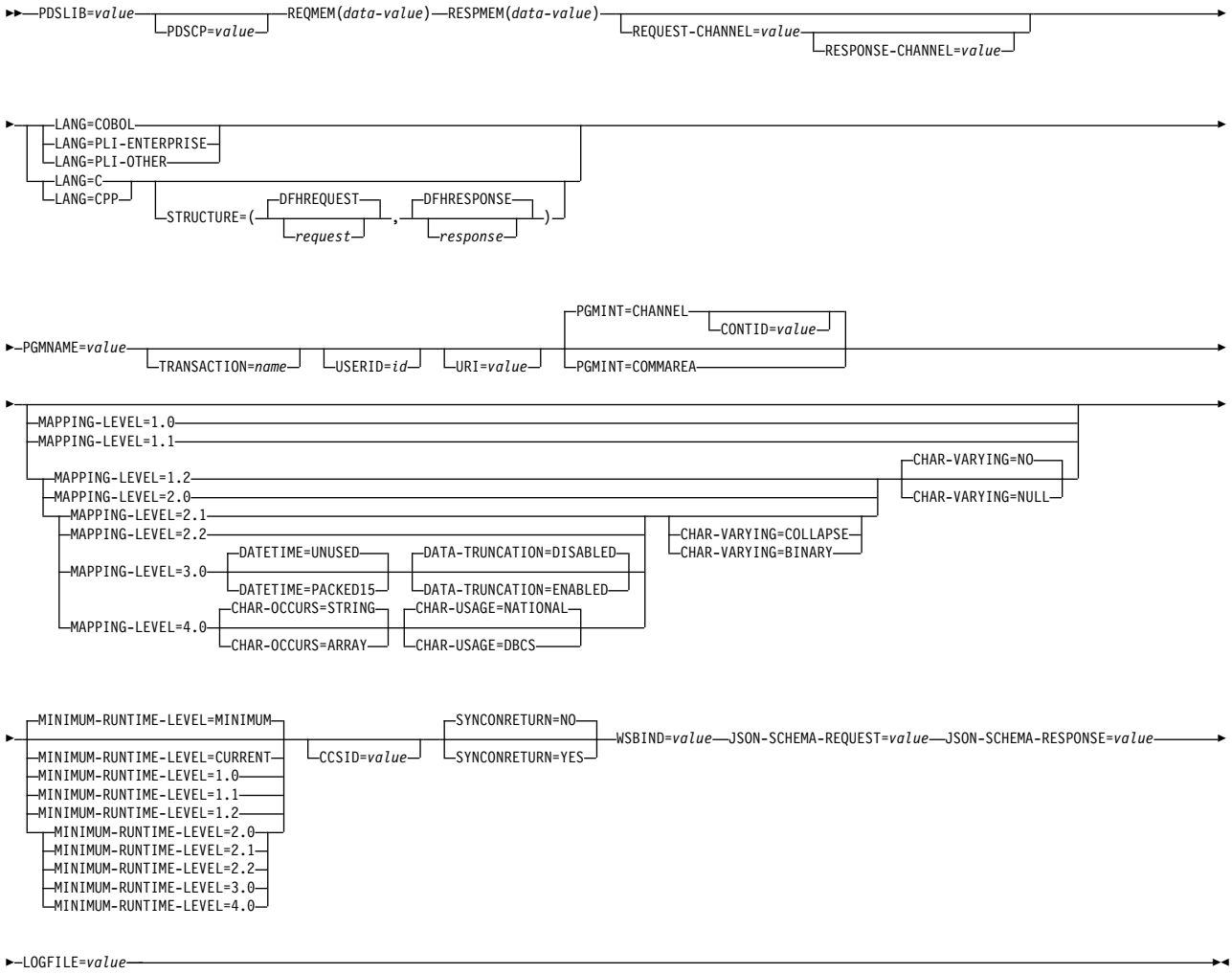
```
/tmp/LS2JS.in
/tmp/LS2JS.out
```

```
/tmp/LS2JS.err
```

Important: DFHLS2JS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHLS2JS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

Therefore, you are advised to devise a naming convention, and operating procedures, that 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 DFHLS2JS



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 - 80 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 Unicode Services User's Guide and Reference. If you do not specify this parameter, the application data structure is encoded by using the CCSID specified in the system initialization parameter.

CHAR-VARYING={**NO**|**NULL**|**COLLAPSE**|**BINARY**}

Specifies how character fields in the language structure are mapped when the mapping level is 1.2 or higher. 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. You can select these options:

NO Character fields are mapped to a JSON string and are processed as fixed-length fields. The maximum length of the data is equal to the length of the field. NO is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping levels 2.0 and earlier.

This value does not apply to Enterprise and Other PL/I language structures.

NULL Character fields are mapped to a JSON string and are processed as null-terminated strings. CICS adds a terminating null character when transforming from a JSON message. The maximum length of the character string is calculated as one character less than the length indicated in the language structure. NULL is the default value for the **CHAR-VARYING** parameter for C/C++.

This value does not apply to Enterprise and Other PL/I language structures.

COLLAPSE

Character fields are mapped to a JSON string. Trailing white space in the field is not included in the JSON message. The inbound JSON message is parsed to remove all leading, trailing, and embedded white space. COLLAPSE is the default value for the **CHAR-VARYING** parameter for COBOL and PL/I at mapping level 2.1 onwards.

BINARY

Character fields are mapped to a JSON string containing base64

encoded data and are processed as fixed-length fields. The BINARY value on the **CHAR-VARYING** parameter is available only at mapping levels 2.1 and onwards.

CHAR-OCCURS={**STRING**|**ARRAY**}

Specifies how character arrays in the language structure are mapped when the mapping level is 4.0 or higher. For example, `PIC X OCCURS 20`. This parameter is only for use by COBOL language.

ARRAY
Character arrays are mapped to a JSON array. This means that every character is mapped as an individual JSON element. This is also the behaviour at mapping levels 3.0 and earlier.

STRING
Character arrays are mapped to an JSON string. This means that the entire COBOL array is mapped as a single JSON element.

CHAR-USAGE={**NATIONAL**|**DBCS**}

In COBOL, the national data type, `PIC N`, can be used for UTF-16 or DBCS data. This setting is controlled by the `NSYMBOL` compiler option. You must set the **CHAR-USAGE** parameter on the assistant to the same value as the `NSYMBOL` compiler option to ensure that the data is handled appropriately. This is typically set to `CHAR-USAGE=NATIONAL` when you use UTF-16.

DBCS Data from `PIC (n)` fields is treated as UTF-16 encoded data.

NATIONAL
Data from `PIC (n)` fields is treated as DBCS encoded data.

CONTID=*value*

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

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}

Specifies if variable length data is tolerated in a fixed-length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**UNUSED**|**PACKED15**}

Specifies if potential ABSTIME fields in the high-level language structure are mapped as timestamps:

PACKED15
Packed decimal fields of length 15 (8 bytes) are treated as CICS ABSTIME fields, and mapped as timestamps.

UNUSED
Packed decimal fields of length 15 (8 bytes) are not treated as timestamps.

You can set this parameter at a mapping level of 3.0.

JSON-SCHEMA-REQUEST=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the request JSON schema is stored.

JSON-SCHEMA-RESPONSE=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the response JSON schema is stored.

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 z/OS UNIX name of the file into which DFHLS2JS writes its activity log and trace information. DFHLS2JS creates the file, but not the directory structure, if it does not exist.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHLS2JS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHLS2JS uses when you generate the web service binding file and JSON schema. You can select these options:

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 Use this mapping level to generate JSON schema using the full set of options available.

4.0 Use this mapping level with a CICS TS 5.2 region. At this mapping level you can use COBOL `OCCURS DEPENDING ON` fields and the **CHAR-OCCURS** parameter.

For more information about mapping levels, see Mapping levels for the CICS JSON assistants.

MINIMUM-RUNTIME-LEVEL={**MINIMUM**|**1.0**|**1.1**|**1.2**|**2.0**|**2.1**|**2.2**|**3.0**|**4.0**|**CURRENT**}

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

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

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 The generated web service binding file deploys into a CICS TS 4.1 region or later.

Note: JSON support is only available from CICS TS 4.2 onwards.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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 that are used for the request and response are specified in the **REQMEM** and **RESPMEM** parameters.

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

PDSCP=*value*
Specifies the code page that is used in the partitioned data set members that are 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, the z/OS UNIX System Services code page is used. For example, you might 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.

- In mapping levels earlier than 3.0, the channel can contain only one container, which is used for both input and output. Use the **CONTID** parameter to specify the name of the container. The default name is DFHWS-DATA.
- At mapping level 3.0, the channel can contain multiple containers. Use the **REQUEST-CHANNEL** and **RESPONSE-CHANNEL** parameters. Do not specify **PDSLIB**, **REQMEM**, or **RESPMEM**.

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

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of the CICS PROGRAM resource for the target application program that is exposed as a web service. The CICS web service support links to this program.

REQMEM=*value*
Specifies the name of the partitioned data set member that 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.

REQUEST-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when it receives a JSON message from a web service requester. The channel description is an XML document that must conform to the CICS supplied channel schema. For more information, see "Creating a channel description document" on page 187.

You can use this parameter at mapping level 3.0 only.

RESPMEM=*value*
Specifies the name of the partitioned data set member that 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.

RESPONSE-CHANNEL=*value*
Specifies the name and location of a channel description document. The channel description describes the containers that the web service provider application can use in its interface when it sends a JSON response message to a web service requester. The channel description is an XML document that must conform to the CICS supplied channel schema. For more information, see "Creating a channel description document" on page 187.

You can use this parameter at mapping level 3.0 only.

STRUCTURE=(*request*,*response*)
For C and C++ only, specifies the names of the high-level structures that are contained in the partitioned data set members that are specified in the **REQMEM** and **RESPMEM** parameters:

request
Specifies the name of the high-level structure that contains 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 that contains 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 sync point.

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

YES The remote web service can issue a sync point. 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 one to four 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ # _ < >

URI=*value*
This parameter specifies the relative or absolute URI that a client uses to access the web service. CICS uses the value that is specified when it generates a

URIMAP resource from the web service binding file that is created by DFHLS2JS. 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 one to eight 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ #

WSBIND=*value*
The fully qualified z/OS UNIX name of the web service binding file. DFHLS2JS creates the file, but not the directory structure, if it does not exist. The file extension is .wsbind.

Other information

- The user ID that DFHLS2JS uses to run must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX file structure and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **JSON Schema** parameters.
- The user ID must have a sufficiently large storage allocation to run Java.
- The JCL has a maximum parameter length of 100 characters. This parameter length can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

Example

```
//LS2JS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHLS2JS,
// TMPFILE=&amp;amp;QT.&SYSUID.&QT,
//INPUT.SYSUT1 DD *
PDSLIB=CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
JSON-SCHEMA-REQUEST=/u/exampleapp/json/example_request.json
JSON-SCHEMA-RESPONSE=/u/exampleapp/json/example_response.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=COLLAPSE
PGMNAME=DFH0XCMN
URI=http://myserver.example.org:8080/exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

DFHJS2LS: JSON schema to high-level language conversion for request-response services

The DFHJS2LS procedure generates a high-level language data structure and a web service binding file from a JSON schema. You can use DFHJS2LS when you prepare to create a CICS application program as a service provider.

The job control statements for DFHJS2LS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

The DFHJS2LS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

The relevant usage mode for the DFHLS2JS or DFHJS2LS procedure depends on your requirements:

- DFHLS2JS: High-level language to JSON schema conversion for linkable interface
- DFHJS2LS: JSON schema to high-level language conversion for linkable interface
- DFHLS2JS: High-level language to JSON schema conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for RESTful services

Job control statements for DFHJS2LS

JOB Starts the job.

EXEC Specifies the procedure name (DFHJS2LS).

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 DFHJS2LS:

JAVADIR=*path*
Specifies the name of the Java directory that is used by DFHJS2LS. The value of this parameter is appended to /usr/lpp/ to produce a complete path name of /usr/lpp/*path*.

Typically, you do not 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 a prefix that extends the z/OS UNIX directory path that is used on other parameters, or '' (empty string) if no prefix is used.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHPREF** parameter.

SERVICE=*value*
Use this parameter only when directed to do so by IBM Support.

TMPDIR=*tmpdir*
Specifies the location of a directory in z/OS UNIX that DFHJS2LS uses as a temporary workspace. 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 DFHJS2LS uses to construct the names of the temporary workspace files.

The default value is JS2LS.

USSDIR=*path*
Specifies the name of the CICS TS directory in the UNIX System Services file system. The value of this parameter is appended to /usr/lpp/cicsts/ to produce a complete path name of /usr/lpp/cicsts/*path*. This must be specified as '.' (period) if the default is used.

Typically, you do not 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

DFHJS2LS creates the following three temporary files at run time:

tmpdir/*tmpprefix*.in

tmpdir/*tmpprefix*.out

tmpdir/*tmpprefix*.err

where:

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

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

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

/tmp/JS2LS.in

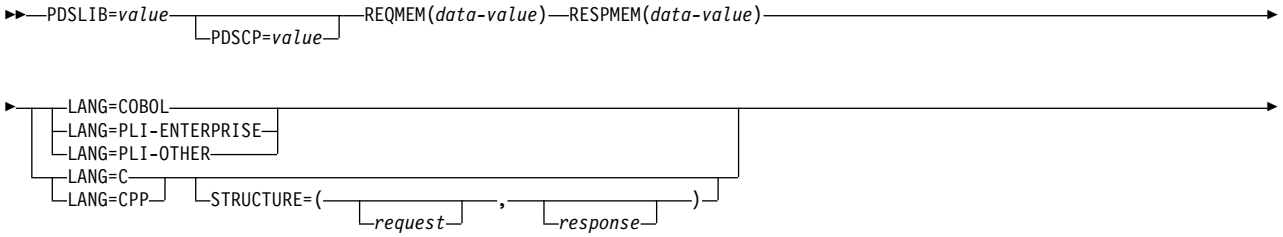
/tmp/JS2LS.out

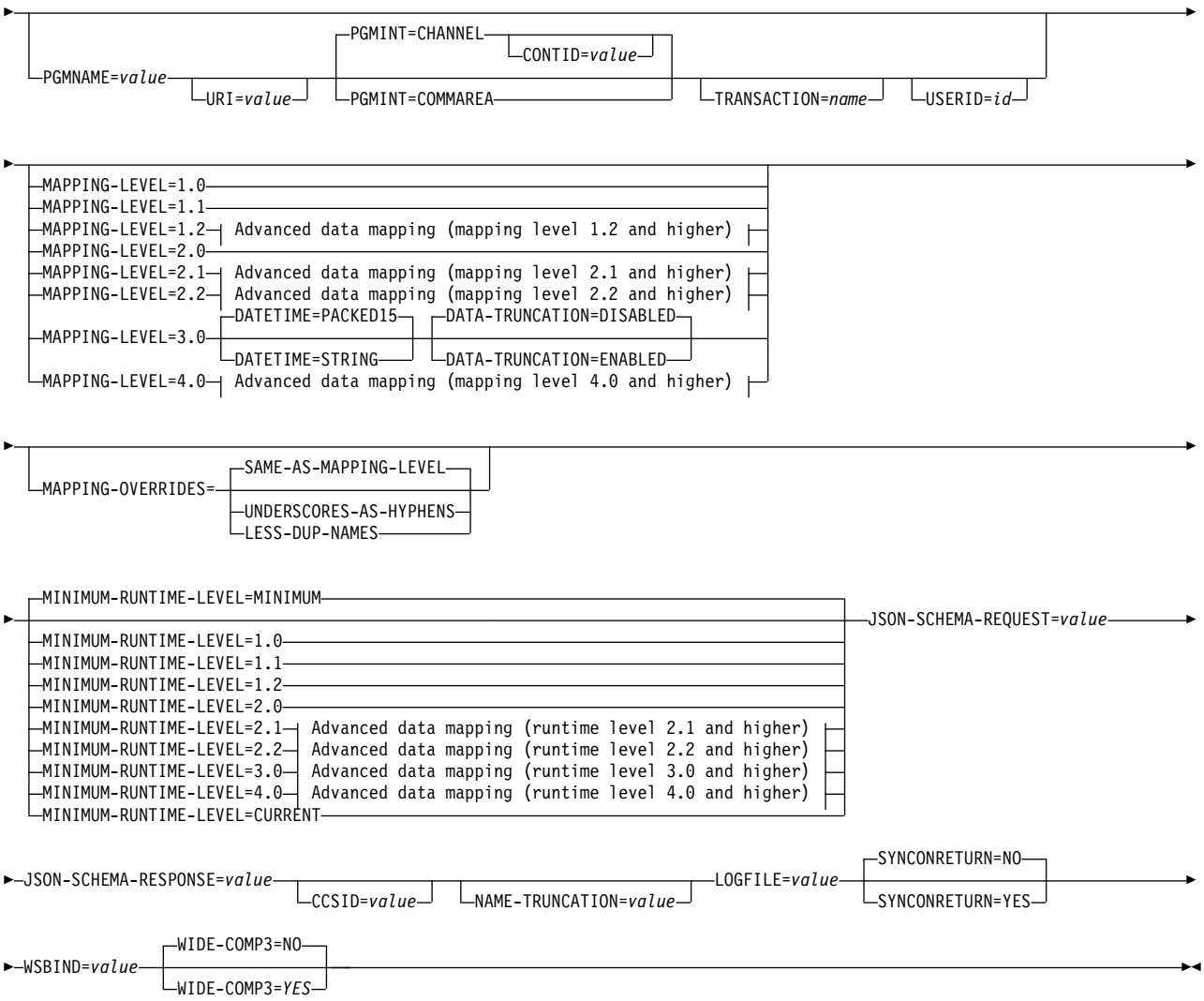
/tmp/JS2LS.err

Important: DFHJS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHJS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

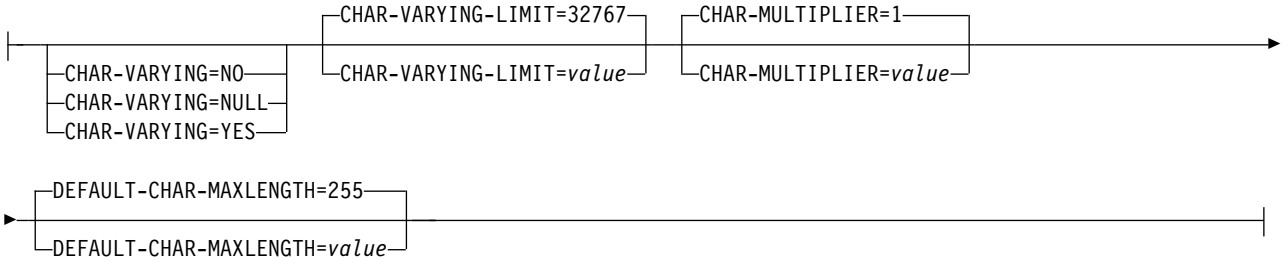
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of the job.

Input parameters for DFHJS2LS

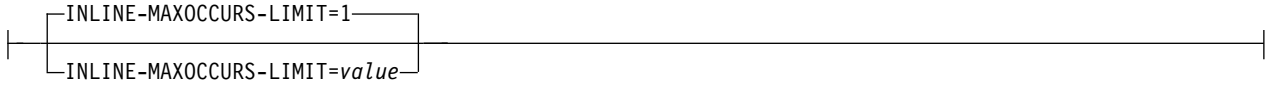




Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):



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 - 80 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 Unicode Services User's Guide and Reference. If you do not specify this parameter, the application data structure is encoded by using the CCSID specified in the system initialization parameter.

CHAR-MULTIPLIER={**1**|*value*}

Specifies the number of bytes to allow for each character when the mapping level is 1.2 or later. The *value* of this parameter can be a positive integer in the range of 1 - 2,147,483,647. 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 might 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.

When you set **CCSID=1200** (indicating UTF-16), the only valid values for **CHAR-MULTIPLIER** are 2 or 4. When you use UTF-16, the default value is 2. Use **CHAR-MULTIPLIER=2** when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use **CHAR-MULTIPLIER=4** when you expect application data to contain characters that require 2 UTF-16 encoding units.

Note: Setting **CHAR-MULTIPLIER** to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger **CHAR-MULTIPLIER** value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.

CHAR-VARYING={**NO**|**NULL**|**YES**}

Specifies how variable-length character data is mapped when the mapping level is 1.2 or higher. Variable-length binary data types are always mapped to

either a container or a varying structure. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

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 two related elements: data-length and the data.

CHAR-VARYING-LIMIT={**32767**|*value*}
Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure when the mapping level is 1.2 or higher. If the character or binary 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 32,767 bytes.

CONTID=*value*
In a service provider, specifies the name of the container that holds the top-level data structure that is used to represent a JSON message.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed-length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME={**PACKED15**|**STRING**}
Specifies how JSON date-time elements are mapped to the language structure.

PACKED15
The default is that any JSON date-time element is processed as a timestamp and is mapped to CICS ABSTIME format.

STRING
The JSON date-time element is processed as text.

DEFAULT-CHAR-MAXLENGTH={**255**|*value*}
Specifies the default array length of character data in characters for mappings where no length is implied in the web service description document, when the mapping level is 1.2 or higher. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647.

INLINE-MAXOCCURS-LIMIT={**1**|*value*}
Specifies whether inline variable repeating content is used based on the `maxItems` JSON schema keyword. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores

each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.

The **INLINE-MAXOCCURS-LIMIT** parameter is available only at mapping level 2.1 onwards. The value of **INLINE-MAXOCCURS-LIMIT** can be a positive integer in the range of 0 - 32,767. A value of `0` indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the *value* of the `maxOccurs` attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise inline mapping is used.

When deciding whether you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.

JSON-SCHEMA-REQUEST=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the request JSON schema is stored.

JSON-SCHEMA-RESPONSE=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the response JSON schema is stored.

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 z/OS UNIX name of the file into which DFHJS2LS writes its activity log and trace information. DFHJS2LS creates the file, but not the directory structure, if it does not exist.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHJS2LS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHJS2LS uses when generating the web service binding file and language structure. You can select these options:

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 Use this mapping level to generate JSON schema using the full set of options available.

4.0 Use this mapping level with a CICS TS 5.2 region when you want to use UTF-16.

For more information about mapping levels, see Mapping levels for the CICS assistants

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|LESS-DUP-NAMES}

Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

LESS-DUP-NAMES

This parameter generates non-structural structure field names with `_value` at the end of the name to enable direct referencing to the field. For example, in the following PLI language structure, when MAPPING-OVERRIDES=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with `_value`:

```
09 streetName,
    12 streetName                    CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

The resulting structure is as follows:

```
09 streetName,
    12 streetName_value              CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

SAME-AS-MAPPING-LEVEL

This parameter generates language structures in the same style as the mapping level. This is the default.

UNDERSCORES-AS-HYPHENS

For COBOL only. This parameter converts any underscores in the JSON schema to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure they are unique. For more information, see JSON schema to COBOL mapping in Developing applications.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}
Specifies the minimum CICS runtime environment into which the web service binding file can be deployed. If you select a level that does not match the other parameters that you specified, you receive an error message. You can select these options:

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

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 The generated web service binding file deploys into a CICS TS 4.1 region or later.

Note: JSON support is only available from CICS TS 4.2 onwards.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

NAME-TRUNCATION={LEFT|RIGHT}
Specifies whether JSON names are truncated from the left or the right. The CICS web services assistant truncates JSON names to the appropriate length for the high-level language specified; by default names are truncated from the right.

PDSCP=*value*
Specifies the code page that is used in the partitioned data set members that are 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, the z/OS UNIX System Services code page is used. For example, you might specify **PDSCP**=037.

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

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.

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*

Specifies the name of a CICS PROGRAM resource.

When DFHJS2LS is used to generate a web service binding file that is used in a service provider, you must supply this parameter. It specifies the resource name of the application program that is exposed as a web service.

When DFHJS2LS is used to generate a web service binding file that is used in a service requester, omit this parameter.

REQMEM=*value*

Specifies a one to six character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contains the high-level language structures for the web service request, which is the input data to the application program.

DFHJS2LS generates the name of partitioned data set member by appending 01 to the prefix.

RESPMEM=*value*

Specifies a one to six character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contains the high-level language structures for the web service response, which is the output data from the application program.

DFHJS2LS generates the name of partitioned data set member by appending 01 to the prefix.

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 *request01* and *response01*.
If one or both names are 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 sync point.

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

YES The remote web service can issue a sync point. 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 one to four 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 by 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 uses to access the web service. CICS uses the value that is specified when it generates a URIMAP resource from the web service binding file that is created by DFHJS2LS. 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 one to eight 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ #

WIDE-COMP3={**NO**|**YES**}
For COBOL only. Controls the maximum size of the packed decimal variable length in the COBOL language structure.

NO DFHJS2LS limits the packed decimal variable length to 18 when generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.

YES DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

WSBIND=*value*
The fully qualified z/OS UNIX name of the web service binding file. DFHJS2LS creates the file, but not the directory structure, if it does not exist. The file extension defaults to **.wsbind**.

Other information

- The user ID under which DFHJS2LS runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS

UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

Example

```
//JS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
JSON-SCHEMA-REQUEST=example.json
JSON-SCHEMA-RESPONSE=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

DFHJS2LS: JSON schema to high-level language conversion for RESTful services

The DFHJS2LS procedure generates a high-level language data structure and a web service binding file from a JSON schema. You can use DFHJS2LS when you prepare to create a RESTful JSON service provider.

The job control statements for DFHJS2LS, its symbolic parameters, its input parameters and their descriptions, and an example job help you to use this procedure.

The DFHJS2LS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

The relevant usage mode for the DFHLS2JS or DFHJS2LS procedure depends on your requirements:

- DFHLS2JS: High-level language to JSON schema conversion for linkable interface
- DFHJS2LS: JSON schema to high-level language conversion for linkable interface
- DFHLS2JS: High-level language to JSON schema conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for request-response services
- DFHJS2LS: JSON schema to high-level language conversion for RESTful services

Job control statements for DFHJS2LS

JOB Starts the job.

EXEC Specifies the procedure name (DFHJS2LS).

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 DFHJS2LS:

JAVADIR=*path*
Specifies the name of the Java directory that is used by DFHJS2LS. The value of this parameter is appended to `/usr/lpp/` to produce a complete path name of `/usr/lpp/`*path*.

Typically, you do not 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 a prefix that extends the z/OS UNIX directory path that is used on other parameters, or `''` (empty string) if no prefix is used.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHPREF** parameter.

SERVICE=*value*
Use this parameter only when directed to do so by IBM Support.

TMPDIR=*tmpdir*
Specifies the location of a directory in z/OS UNIX that DFHJS2LS uses as a temporary workspace. 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 DFHJS2LS uses to construct the names of the temporary workspace files.

The default value is `JS2LS`.

USSDIR=*path*
Specifies the name of the CICS TS directory in the UNIX System Services file system. The value of this parameter is appended to `/usr/lpp/cicsts/` to produce a complete path name of `/usr/lpp/cicsts/`*path*. This must be specified as `'.'` (period) if the default is used.

Typically, you do not 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

DFHJS2LS creates the following three temporary files at run time:

tmpdir`/`*tmpprefix*`.in`

tmpdir`/`*tmpprefix*`.out`

tmpdir`/`*tmpprefix*`.err`

where:

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

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

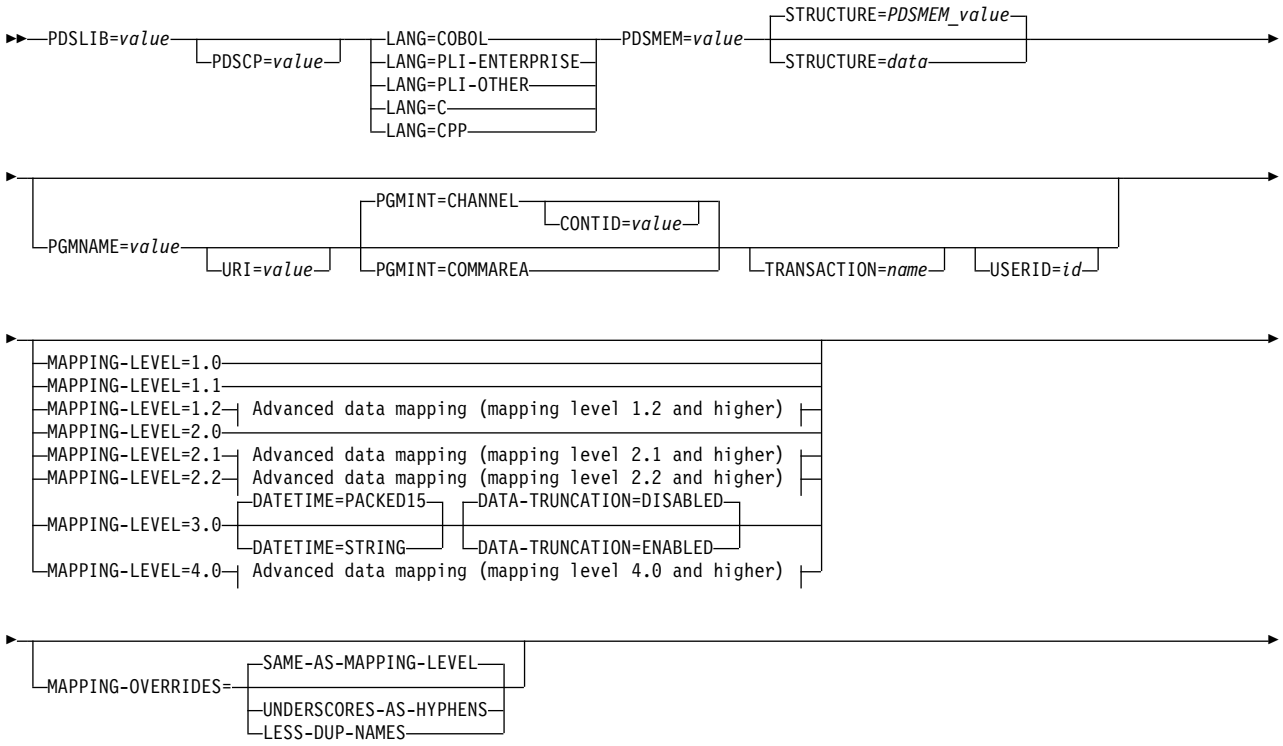
The default names for the files, when **TMPDIR** and **TMPFILE** are not specified, are as follows:

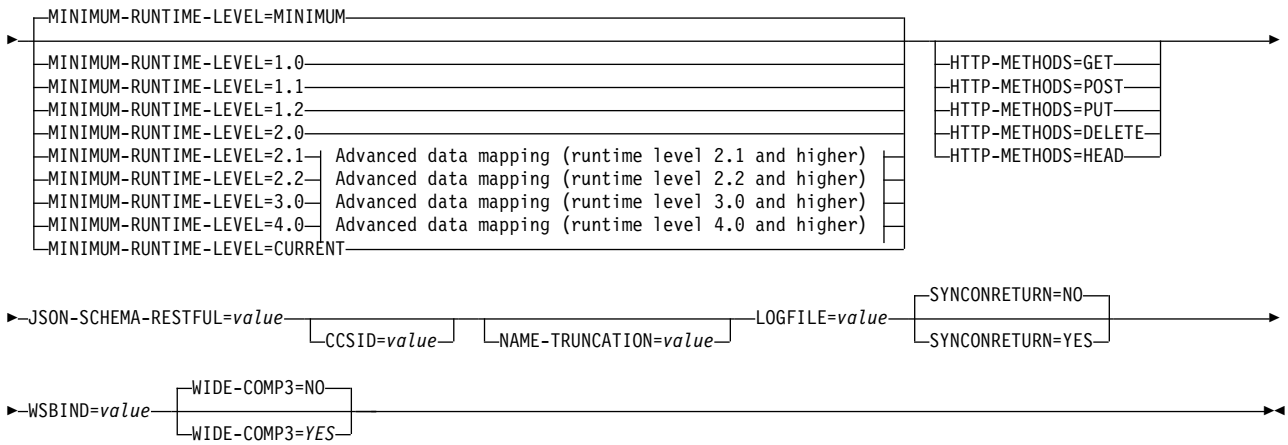
```
/tmp/JS2LS.in
/tmp/JS2LS.out
/tmp/JS2LS.err
```

Important: DFHJS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHJS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

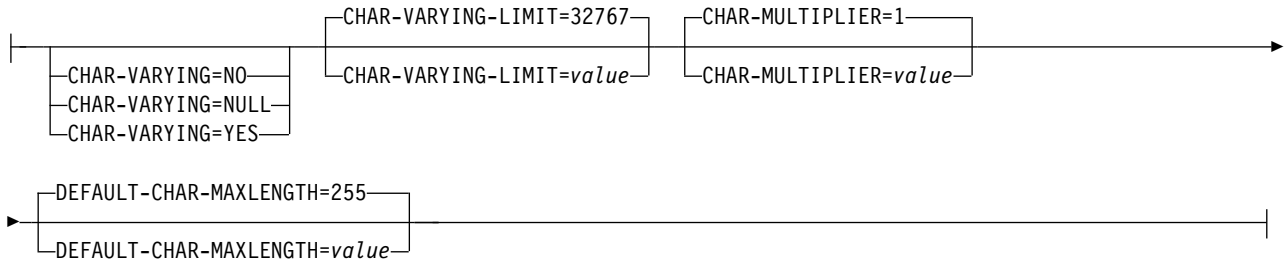
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of the job.

Input parameters for DFHJS2LS

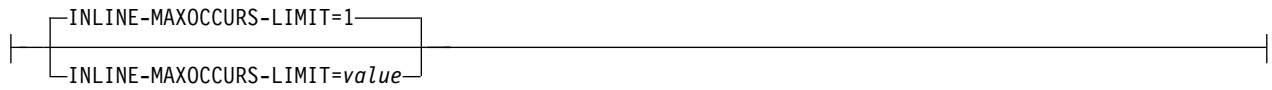




Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):



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 - 80 must 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.
- A comma in the last character position of the line is an optional line separator, and 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 Unicode Services User's Guide and Reference. If you do not specify this parameter, the application data structure is encoded by using the CCSID specified in the system initialization parameter.

CHAR-MULTIPLIER={**1**|*value*}
Specifies the number of bytes to allow for each character when the mapping level is 1.2 or later. The *value* of this parameter can be a positive integer in the range of 1 - 2,147,483,647. 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 might 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.

When you set **CCSID=1200** (indicating UTF-16), the only valid values for **CHAR-MULTIPLIER** are 2 or 4. When you use UTF-16, the default value is 2. Use **CHAR-MULTIPLIER=2** when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use **CHAR-MULTIPLIER=4** when you expect application data to contain characters that require 2 UTF-16 encoding units.

Note: Setting **CHAR-MULTIPLIER** to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger **CHAR-MULTIPLIER** value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.

CHAR-VARYING=**NO**|**NULL**|**YES**
Specifies how variable-length character data is mapped when the mapping level is 1.2 or higher. Variable-length binary data types are always mapped to either a container or a varying structure. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

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 two related elements: data-length and the data.

CHAR-VARYING-LIMIT=**32767**|*value*
Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure when the mapping level is 1.2 or higher. If the character or binary 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 32,767 bytes.

CONTID=*value*
In a service provider, specifies the name of the container that holds the top-level data structure that is used to represent a JSON message.

The length of the container that CICS passes to the target application program is the greater of the lengths of the request container and the response container.

DATA-TRUNCATION={**DISABLED**|**ENABLED**}
Specifies if variable length data is tolerated in a fixed-length field structure:

DISABLED
If the data is less than the fixed length that CICS is expecting, CICS rejects the truncated data and issues an error message.

ENABLED
If the data is less than the fixed length that CICS is expecting, CICS tolerates the truncated data and processes the missing data as null values.

DATETIME=**PACKED15**|**STRING**
Specifies how JSON date-time elements are mapped to the language structure.

PACKED15
The default is that any JSON date-time element is processed as a timestamp and is mapped to CICS ABSTIME format.

STRING
The JSON date-time element is processed as text.

DEFAULT-CHAR-MAXLENGTH=**255**|*value*
Specifies the default array length of character data in characters for mappings where no length is implied in the web service description document, when the mapping level is 1.2 or higher. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647.

HTTP-METHODS=**{***GET*|*POST*|*PUT*|*DELETE*|*HEAD***},{***GET*|*POST*|*PUT*|*DELETE*|*HEAD***},***
This is an optional parameter.

The default value is for GET, POST, PUT, and DELETE to be set, which tells DFHJS2LS that the application supports the four main RESTful operations.
If a value is provided, DFHJS2LS builds a WSBind file in which only the explicitly specified HTTP methods are accepted.
If an application wants to implement the HEAD method, it must deliberately opt-in to doing so. By default DFHJS2LS assumes that the application does not support incoming HTTP HEAD methods.
If a JSON client attempts to use an unsupported HTTP method, CICS responds with an HTTP 405 response.

INLINE-MAXOCCURS-LIMIT=**1**|*value*
Specifies whether inline variable repeating content is used based on the `maxItems` JSON schema keyword. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.

The **INLINE-MAXOCCURS-LIMIT** parameter is available only at mapping level 2.1 onwards. The value of **INLINE-MAXOCCURS-LIMIT** can be a positive integer in the range of 0 - 32,767. A value of `0` indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the *value* of the `maxOccurs` attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise inline mapping is used.

When deciding whether you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.

JSON-SCHEMA-RESTFUL=*value*
This is a mandatory parameter.

The value indicates the UNIX System Services location where the response JSON schema is stored.

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 z/OS UNIX name of the file into which DFHJS2LS writes its activity log and trace information. DFHJS2LS creates the file, but not the directory structure, if it does not exist.

Typically, you do not use this file, but it might be requested by the IBM service organization if you encounter problems with DFHJS2LS.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}
Specifies the level of mapping that DFHJS2LS uses when generating the web service binding file and language structure. You can select these options:

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 Use this mapping level to generate JSON schema using the full set of options available.

4.0 Use this mapping level with a CICS TS 5.2 region when you want to use UTF-16.

For more information about mapping levels, see Mapping levels for the CICS JSON assistants in Developing applications.

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|LESS-DUP-NAMES}

Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

LESS-DUP-NAMES

This parameter generates non-structural structure field names with `_value` at the end of the name to enable direct referencing to the field. For example, in the following PLI language structure, when MAPPING-OVERRIDES=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with `_value`:

```
09 streetName,
    12 streetName                    CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

The resulting structure is as follows:

```
09 streetName,
    12 streetName_value              CHAR(255) VARYING
   UNALIGNED,
    12 filler                        BIT (7),
    12 attr_nil_streetName_value     BIT (1),
```

SAME-AS-MAPPING-LEVEL

This parameter generates language structures in the same style as the mapping level. This is the default.

UNDERSCORES-AS-HYPHENS

For COBOL only. This parameter converts any underscores in the JSON schema to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure they are unique. For more information, see JSON schema to COBOL mapping in Developing applications.

MINIMUM-RUNTIME-LEVEL={MINIMUM|1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0|CURRENT}

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

MINIMUM

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

1.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

1.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.0 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.1 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

2.2 This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

3.0 The generated web service binding file deploys into a CICS TS 4.1 region or later.

Note: JSON support is only available from CICS TS 4.2 onwards.

4.0 The generated web service binding file deploys successfully into a CICS TS 5.2 region or later. With this runtime level, you can use a mapping level of 4.0 or earlier for the **MAPPING-LEVEL** parameter. 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.

NAME-TRUNCATION={LEFT|RIGHT}
Specifies whether JSON names are truncated from the left or the right. The CICS web services assistant truncates JSON names to the appropriate length for the high-level language specified; by default names are truncated from the right.

PDSCP=*value*
Specifies the code page that is used in the partitioned data set members that are specified in the **PDSMEM** parameter, where *value* is a CCSID number or a Java code page number. If this parameter is not specified, the z/OS UNIX System Services code page is used. For example, you might specify **PDSCP**=037.

PDSLIB=*value*
Specifies the name of the partitioned data set that contains the generated high-level language. The data set members that are used for the request and response are specified in the **PDSMEM** parameter.

PDSMEM=*value*
Specifies the 1-6 character prefix that DFHJS2LS uses to generate the name of the partitioned data set member that will contain the high-level language structures.

DFHJS2LS generates a partitioned data set member by appending 01 to the prefix.

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.

When the target application program has processed the request, it must use the same mechanism to return the response. If the request was received in a communication area, then the response must be returned in the communication area; if the request was received in a container, the response must be returned in a container. The length of the communication area or container that CICS passes to the target application program is the greater of the lengths of the request communication area or container and the response communication area or container.

PGMNAME=*value*
Specifies the name of a CICS PROGRAM resource.

When DFHJS2LS is used to generate a web service binding file that is used in a service provider, you must supply this parameter. It specifies the resource name of the application program that is exposed as a web service.

When DFHJS2LS is used to generate a web service binding file that is used in a service requester, omit this parameter.

STRUCTURE=*name*
For C and C++ only, specifies how the name of the structure is generated.

The generated structure is given the name of *name01* .
If the name is omitted, the structure has the same name as the partitioned data set member name generated from the **PDSMEM** parameter that you specify.

SYNCONRETURN={**NO**|**YES**}
Specifies whether the remote web service can issue a sync point.

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

YES The remote web service can issue a sync point. 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 one to four 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 by 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 uses to access the web service. CICS uses the value that is specified when it generates a URIMAP resource from the web service binding file that is created by DFHJS2LS. The parameter specifies the path component of the URI to which the URIMAP definition applies. When using wildcards * at the end of a URI, the URI value must be followed by a comma.

USERID=*id*

In a service provider, this parameter specifies a one to eight 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 by using the **PIPELINE** scan command.

Acceptable characters:
A-Z a-z 0-9 $ @ #

WIDE-COMP3=**NO**|**YES**

For COBOL only. Controls the maximum size of the packed decimal variable length in the COBOL language structure.

NO DFHJS2LS limits the packed decimal variable length to 18 when generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.

YES DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

WSBIND=*value*

The fully qualified z/OS UNIX name of the web service binding file. DFHJS2LS creates the file, but not the directory structure, if it does not exist. The file extension defaults to .wsbind.

Other information

- The user ID under which DFHJS2LS runs must be configured to use UNIX System Services. The user ID must have read permission to the CICS z/OS UNIX 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.
- The JCL has a maximum parameter length of 100 characters. This can be increased by using the **STDPARM** statement, for more information, see *z/OS UNIX System Services User Guide*.

Example

```
//JS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA-RESTFUL=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/wsbind/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
INLINE-MAXOCCURS-LIMIT=2
PGMNAME=DFH0XCMN
URI=exampleApp/example/*,
PGMINT=COMMAREA
SYNCONRETURN=YES
WSBIND=/u/exampleapp/wsbind/example.wsbind
/*
```

Creating a JSON service provider application by using the JSON assistant

You can create a service provider application from a JSON schema that complies with JSON schema v4 (draft), or from a high-level language data structure. The CICS JSON assistant helps you to deploy your CICS applications in a service provider setting.

About this task

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

- Start with a JSON schema and use the assistant to generate the language data structures.

  Use this option when you want to implement a service provider that conforms with an existing web service description.
- Start with the language data structures and use the assistant to generate the JSON schema.

  Use this option when you want to expose an existing program as a JSON service.

Related information:

JSON schema v4 (draft)

Creating a service provider application from a JSON schema

Using the CICS JSON assistant, you can create a service provider application from a JSON schema.

Before you begin

Before you can create a service provider application, the following conditions must be satisfied:

- Your web services description must be in a UNIX file in z/OS and you must create a suitable provider mode pipeline in the CICS region.
- You must define to OMVS the user ID under which DFHJS2LS runs.
- The user ID must have read permission to z/OS UNIX and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **JSON-SCHEMA-REQUEST** , **JSON-SCHEMA-RESPONSE**, and **JSON-SCHEMA-RESTFUL** parameters.
- You must allocate sufficient storage to the user ID for the ID to run Java. You can use any supported version of Java. By default, DFHJS2LS uses the Java version specified in the **JAVADIR** parameter.

About this task

You can use the JSON assistant to create language structures from your JSON schema for the service provider application.

Procedure

1. Use the DFHJS2LS batch program to generate a web service binding file and one or more language data structures. DFHJS2LS contains a large set of optional parameters that provide you with flexibility to create the binding file

and language structures that your application requires. Consider these options when you enable an existing application for web services:

- Which mechanism will CICS use to pass data to the service provider application program? You can use channels and pass the data in containers or use a COMMAREA. Channels and containers are recommended. Specify them with the **PGMINT** parameter.
- Which language do you want to generate? DFHJS2LS can generate COBOL, C/C++, or PL/I language data structures. Specify the language using the **LANG** parameter.
- Which mapping level do you want to use? The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You are recommended to use the highest level of mapping available. Specify the mapping level with the **MAPPING-LEVEL** parameter.
- Which URI do you want the web service requester to use? Specify a relative URI using the **URI** parameter; for example, `URI=/my/test/webservice`. The value is used by CICS when it creates the URIMAP resource.
- Under which transaction and user ID will you run the web service request and response? You can use an alias transaction to run the application to compose a response to the service requester. The alias transaction is attached under the user ID. Specify it with the **TRANSACTION** and **USERID** parameters. These values are used when creating the URIMAP resource. If you do not want to use a specific transaction, do not use these parameters.

When you submit DFHJS2LS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The language structures are placed in the partitioned data set that you specified with the **PDSLIB** parameter.

2. Copy the generated web service binding file to the `pickup` directory of the provider mode PIPELINE resource that you want to use for your web service application. You must copy the binding file in binary mode.
3. Write a service provider application program to interface with the generated language structures and implement the required business logic.
4. Use the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and a URIMAP resources.
   - The WEBSERVICE resource encapsulates the web service binding file in CICS and is used at run time.
   - The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.

   Alternatively, you can define the resources yourself, although this is not recommended.

Results

If you have any problems submitting DFHJS2LS, or the resources do not install correctly, see Troubleshooting the JSON assistant in Troubleshooting.

JSON schema v4

JSON schema validation v0

Creating a service provider application from a data structure

Use the CICS web services assistant to create a service provider application from a high-level language data structure.

Before you begin

Before you create a service provider application, make sure that your setup complies with these preconditions:

- Your high-level language data structures must meet the following criteria:
  - The data structures must be defined separately from the source program; for example, in a COBOL copybook.
  - If your PL/I or COBOL application program uses different data structures for input and output, the data structures must be defined in two different members in a partitioned data set. If the same structure is used for input and output, the structure must be defined in a single member.

    For C and C++, your data structures can be in the same member in a partitioned data set.
- The language structures must be available in a partitioned data set and you must create a suitable PIPELINE resource in the CICS region.
- You must define to OMVS the user ID that DFHLS2JS uses to run.
- The user ID must have read permission to z/OS UNIX and PDS libraries and write permission to the directories specified on the **LOGFILE**, **WSBIND**, and **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** output parameters.
- The user ID must have a sufficiently large storage allocation to run Java. You can use any supported version of Java. By default, DFHLS2JS uses the Java version that is specified in the **JAVADIR** parameter.

Procedure

Follow these steps to create a service provider application from a high-level data structure:

1. If the service provider application interface uses channels and many containers, create a channel description document that describes the interface in JSON. You must save the channel description document in a suitable directory on z/OS UNIX. CICS uses this document to construct and deconstruct a JSON message from the containers on a channel. Alternatively, you can use one container in a channel and not create a channel description document.

   For more information about how to create a channel description document, see "Creating a channel description document" on page 187.
2. Use the DFHLS2JS batch program to generate a web service binding file and web service description from the language structure. The DFHLS2JS batch program can be found in *HLQ*.XDFHINST where *HLQ* is the location where you installed CICS. DFHLS2JS contains a large set of optional parameters that provide you with flexibility to create the binding file and language structures that your application requires. Consider the following options when enabling web services for an existing application:
   - Which mechanism do you want CICS to use to pass data to the service provider application program? You can use channels and pass the data in containers, or use a COMMAREA. Specify the mechanism by using the **PGMINT** parameter. If your application interface uses channels and many

containers, specify the **REQUEST-CHANNEL** parameter and optionally the **RESPONSE-CHANNEL**. You can use these parameters only when the mapping level is 3.0 or higher.

- Which mapping level do you want to use? The higher the mapping level, the more control and support you have available for the handling of character and binary data at run time. Some optional parameters are available only at the higher mapping levels. You must specify the highest level of mapping available in the **MAPPING-LEVEL** parameter.
- Which URI do you want the web service to use? Specify an absolute URI by using the **URI** parameter; for example, **URI**=`http://www.example.org:80/my/test/webservice`. The relative part of this address, `/my/test/webservice`, is used when creating the URIMAP resource.

When you submit DFHLS2JS, CICS generates the web service binding file and places it in the location that you specified with the **WSBIND** parameter. The generated JSON schemas are placed in the location that you specified with the **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** parameters.

3. Review the generated JSON schema.

   These schemas are used to define the input and output data formats for the JSON web service. The application developer must use these schemas when creating an application to interact with the JSON web service.

   Note: Changing the generated schema invalidates the generated web service binding file, `WSBind`.

   If you want to change the schema, for example, to rename the fields within the schema, you must use DFHJS2LS to generate a new web service binding file, and a new set of language structures. The application program in CICS must be changed to use the new language structures.
4. Copy the web service binding file to the pickup directory of the provider mode pipeline that you want to use for your web service application. You must copy the web service binding file in binary mode.
5. Use the **PIPELINE SCAN** command to dynamically create the WEBSERVICE resource and a URIMAP resource.
   - The WEBSERVICE resource contains the web service binding file in CICS and is used at run time.
   - The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI.

   Alternatively, you can define the resources yourself.

Results

The creation of your service provider application is complete.

If you have any problems submitting DFHLS2JS, or the resources do not install correctly, see Troubleshooting the JSON assistant in Troubleshooting.

What to do next

Make the web services description available to anyone who develops a web service that will access your service.

Creating a channel description document

Create a channel description document when your service provider application uses a channel interface with many containers.

About this task

Use an XML editor to create the channel description document. The schema for the channel description is called `channel.xsd` and is in the `/usr/lpp/cicsts/cicsts53/schemas/channel` directory (where `/usr/lpp/cicsts/cicsts53` is the default install directory for CICS files on z/OS UNIX).

Procedure

1. Create an XML document with a <channel> element and the CICS channel namespace:

   ```
   <channel name="myChannel" xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
   </channel>
   ```

2. Add a <container> element for every container that the application program interface uses on the channel. You must use name, type and use attributes to describe each container. The following example shows six containers with different attribute values:

   ```
   <container name="cont1" type="char" use="required"/>
   <container name="cont2" type="char" use="optional"/>
   <container name="cont3" type="bit" use="required"/>
   <container name="cont4" type="bit" use="optional"/>
   <container name="cont5" type="bit" use="required">
       <structure location="//HLQ.PDSNAME(MEMBER)"/>
   </container>
   <container name="cont6" type="bit" use="optional">
       <structure location="//HLQ.PDSNAME(MEMBER2)"/>
   </container>
   ```

   The structure element indicates that the content is defined in a language structure located in a partitioned data set member.

3. Save the XML document in z/OS UNIX.

Channel schema

The channel description document must conform to the following schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
   xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel" elementFormDefault="qualified">
   <element name="channel"> 1
      <complexType>
         <sequence>
            <element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0"> 2
               <complexType>
                  <sequence>
                     <element name="structure" minOccurs="0"> 3
                       <complexType>
                           <attribute name="location" type="string" use="required"/>
                           <attribute name="structure" type="string" use="optional"/>
                       </complexType>
                     </element>
                  </sequence>
                  <attribute name="name" type="tns:name16Type" use="required"/>
                  <attribute name="type" type="tns:typeType" use="required"/>
                  <attribute name="use" type="tns:useType" use="required"/>
               </complexType>
             </element>
          </sequence>
          <attribute name="name" type="tns:name16Type" use="optional" />
```

```
      </complexType>
   </element>
  <simpleType name="name16Type">
      <restriction base="string">
         <maxLength value="16"/>
      </restriction>
  </simpleType>
  <simpleType name="typeType">
      <restriction base="string">
        <enumeration value="char"/>
        <enumeration value="bit"/>
      </restriction>
  </simpleType>
  <simpleType name="useType">
     <restriction base="string">
        <enumeration value="required"/>
        <enumeration value="optional"/>
     </restriction>
  </simpleType>
</schema>
```

1. This element represents a CICS channel.
2. This element represents a CICS container within the channel.
3. A structure can only be used with 'bit' mode containers. The 'location' attribute indicates the location of a file that maps the contents of container. The 'structure' attribute may be used in C and C++ to indicate the name of structure.

What to do next

Run DFHLS2JS to create the mappings and JSON schema for the web service provider application. DFHLS2JS puts the mappings for the channel in the JSON schema in the order that the containers are specified in the channel description document.

Customizing generated JSON schemas

The JSON schemas that are generated by DFHLS2JS contain some automatically generated content that might be appropriate for you to change before publishing. Customizing JSON schemas can result in regenerating the web services binding file and, in some cases, writing a wrapper program.

About this task

Follow these steps to customize generated JSON schemas:

Procedure

1. To advertise support for HTTPS, use the **URI** parameter in DFHLS2JS to set an absolute URI.
2. To supply the network location of your web service, use the **URI** parameter in DFHLS2JS to set an absolute URI.
3. Consider whether the automatically generated names in the JSON schema are appropriate for your purposes. You can rename the properties.

   These values form part of the programmatic interface to which you code a client program. If the generated names are not sufficiently meaningful, maintenance of your application code might be more difficult over a long period of time.

   If you change any of these values, you must use DFHJS2LS to regenerate the web services binding file. The language structures that are produced are unlikely to be compatible with your existing application, so application changes

might be required. Review the generated language structures, and consider writing a wrapper program as discussed in step 4 on page 189.

4. Consider if the COMMAREA fields exposed in the JSON schemas are appropriate. You might consider removing any fields that are not helpful to a JSON client developer:
   - Fields that are used only for output values can be removed from the schema that maps the input data structures.
   - Filler fields.
   - Automatically generated annotations.

   If you make any of these changes, you must regenerate the web services binding file using DFHJS2LS. The new language structures that are generated are not compatible with the original language structures, so you must write a wrapper program to map data from the new representation to the old one. This wrapper program needs to perform an **EXEC CICS LINK** command 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 JSON client developers.

Results

You have a customized JSON schema that matches your business requirements, and a PROGRAM in CICS that implements them.

Creating a RESTful web service provider application

The CICS implementation of RESTful JSON web services is similar to that of SOAP web services. Most of the concepts and architecture are shared, but CICS requires the use of a JSON schema.

About this task

Implementing a RESTful JSON web service involves the following tasks:

Procedure

1. Generate the application interface.

   Input:
   - The JSON schema that defines the data model for the RESTful web service.

   Output:
   - The language structures (for example, COBOL copy book) that map the JSON schema.
   - A WSBind file.

   Run the DFHJS2LS utility and specify the appropriate input parameters. These parameters include:
   - The location of the JSON schema.
   - The list of supported methods (GET, PUT, POST, and DELETE are enabled by default).
   - The URI at which the service is deployed.
   - The name of the application PROGRAM that implements the service.
   - Any required data-mapping parameters.
2. Create an application that uses this interface.

   Input:

- The language structures from step 1 on page 190.
- An awareness of which RESTful operations will be implemented

Output:

- A program suitable for deployment to CICS

Write a program that does the following:

Note: If you supply the name of your own container on the **CONTID** parameter to DFHJS2LS, you must use this container instead of DFHWS-DATA whenever it is mentioned in the following steps.

a. Examine the URI to understand the identity of the resource. CICS provides several containers to help you identify interesting components of the URI. The containers are described in Table 2 on page 191. The examples show the contents of each container for the URI:

   ```
   http://www.example.org:10000/JSONServices/CustomerDetails/13388?action=query
   ```

   If the URIMAP that matched has PATH of `/JSONServices/CustomerDetails/*`

Table 16. DFHWS-URI containers. DFHWS-URI containers

Container	Contents	Example
DFHWS-URI in Configuring	The complete URI	`http://www.example.org:10000/JSONServices/CustomerDetails/13388?action=query`
	The portion of URI path that matched the URIMAP	`/JSONServices/CustomerDetails/*`
DFHWS-URI-RESID in Configuring	The URI path with the portion matched by the URIMAP removed (*the resource identifier*)	`13388`
DFHWS-URI-QUERY in Configuring	The query string	`action=query`

b. Validate the URI. If there is a problem, report the problem to CICS and terminate.

c. Query the DFHHTTPMETHOD container to determine which method is being driven. For more information, see DFHHTTPMETHOD in Configuring.

d. For a POST (create) method (if needed):
   - Read the input data from the DFHWS-DATA container.
   - Interpret the data by using the language structure(s) generated in step 1 on page 190.
   - Validate the data. If there is a problem, report the problem to CICS and terminate.
   - Perform whatever application-specific processing is required to create the *resource*.
   - Optionally, write to the DFHRESPONSE container to notify the client of the identifier of the new resource. The contents of this container are not transformed by CICS, but sent directly in the HTTP response. The DFHWS-DATA container is ignored.

e. If a GET (inquire) or HEAD (inquire) method is needed, write the data that represents the *resource* to the DFHWS-DATA container.

f. If a PUT (set) method is needed:
   - Read the input data from the DFHWS-DATA container.
   - Interpret the data by using the language structure(s) generated in step 1 on page 190.
   - Validate the data. If there is a problem, report the problem to CICS and terminate.
   - Perform whatever application-specific processing is required to update the *resource*.

g. If a DELETE method is needed, perform whatever application-specific processing is required to delete the *resource*.

Note: The RESTful data model that is implemented by CICS does not send a response body for the PUT, POST, or DELETE methods by default. RESTful applications typically use the HTTP status code to indicate success or failure. If the application completes normally, CICS sends an HTTP response of 200 (OK). For more information about sending error responses, see Application error reporting in Developing applications. If you want to send a response body for a PUT, POST or DELETE method, you must write the DFHRESPONSE container. If present, CICS sends the contents of this container in the HTTP body without further processing. CICS ignores the DFHWS-DATA container in the response processing for these methods.

3. Deploy the artifacts.

   Input:
   - The WSBind file from step 1 on page 190.
   - The CICS Program from step 2 on page 190.
   - A CICS provider mode pipeline resource that is configured for JSON.

   Output:
   - A deployed RESTful JSON web service.

   Deploy the program to CICS in the normal way.

   Either:
   - Deploy the WSBind file to the Pipeline's 'pickup' directory; then issue a **PIPELINE SCAN** command to create the WEBSERVICE and URIMAP resources.
   - Manually define and install a WEBSERVICE resource and associated URIMAP resource. The URIMAP must associate the URI with both the PIPELINE and the WEBSERVICE.

4. Test the service.

   Input:
   - The JSON schema.
   - The URI of the RESTful web service.
   - The deployed service from step 3.
   - A JSON test client of your choice.

   Output:
   - A successfully handled request.

   Use the test client of your choice to send a JSON request to CICS.

   If you receive an unexpected response, attempt problem determination. For more information, see Troubleshooting problems with JSON requests.

Related information:

Design considerations for RESTful web service provider applications

This topic describes some issues you should consider when planning and designing a RESTful web service provider application for JSON.

Collections of Resources

A common design for RESTful APIs is to support the retrieval of collections of resources. For example, a Service might exist that returns a set of objects as follows:

```
GET /Services/CustomerDetails?Surname=Cooper
```

This request is expected to return information on all CustomerDetails objects where the Surname is "Cooper". Individual CustomerDetails objects may be returned using a more specific URI such as:

```
GET /Services/CustomerDetails/Customer27
```

In this example `Customer27` is the primary key for a specific customer. The output from this second query will be an instance of the CustomerDetails object. The output from the first query is less clear: it could return a list of CustomerDetails objects, or it could return a list of URIs for CustomerDetails objects (which the client can go on to retrieve individually). Both conventions are common.

To implement a Collection in CICS, create a JSON schema that describes either a list of data instances, or a list of URIs. You can then build the Service and implement it as normal. In this example you might choose to only implement the GET method. You could consider implementing a pagination Service to allow a client to page backwards and forwards through large data sets, for example:

```
GET /Services/CustomerDetails?startRecord=200&endRecord=225
```

You're likely to need two URIMAP resources in CICS (and two WEBSERVICE resources). One that maps the root URI structure for the Collection, and a wild-carded URIMAP for the Instances. For example:

```
URIMAP1: Path=/Services/CustomerDetails  WEBSERVICE=CollectionService
URIMAP2: Path=/Services/CustomerDetails/* WEBSERVICE=InstanceService
```

Cache Management

The RESTful architecture encourages integration with standard HTTP cache management techniques. This allows the results of GET requests to be cached in the network, thereby reducing the burden on the server. The mechanism for doing this involves setting an expiration date/time for data returned for GET requests.

There is no general purpose mechanism to support application controlled cache expiration in CICS, but a Pipeline Handler program could be written to add the appropriate HTTP Header using the EXEC CICS WEB WRITE HTTPHEADER API. Application programs can do something similar, but only if they are hosted in the same CICS region that receives the HTTP request.

Application error reporting

Read this topic to understand how RESTful JSON web service provider applications can report errors to clients.

In top-down scenarios, it is likely that the application is required to report error conditions. For example, an error might be: "`Account Number not recognized`". This requirement is unique to top-down scenarios; in bottom-up development the application either has an error reporting mechanism that is encoded in the data fields, or it abends. In the top-down scenario, the JSON schema is unlikely to define a field in which to report errors, so an alternative is needed.

For SOAP-based web services, this problem is addressed by using the **EXEC CICS SOAPFAULT** API. SOAP Fault messages do not exist in JSON. Instead, you can use the DFHHTTPSTATUS container to report application detected errors for JSON applications. For more information, see DFHHTTPSTATUS in Configuring.

Note: Applications might also use the DFHRESPONSE container, and other control containers, to provide a more detailed error response, if they want to do so.

JSON web service restrictions

Use this reference material to understand capabilities that are not supported by JSON web services.

The following capabilities are not supported:

- Requester mode pipelines with JSON are not supported.
- Runtime validation of JSON data against schema is not supported. The value of the VALIDATION attribute of a WEBSERVICE resource that is used with a JSON payload is ignored.
- Use of namespaces in JSON data (Badgerfish or Mapped conventions) is not supported.
- JSON payloads sent to CICS must be encoded in UTF-8. No other encoding is supported. Similarly, JSON sent by CICS is always encoded in UTF-8.
- WebSphere MQ transports with JSON pipelines are not supported.
- Vendor transformer programs are not supported for use with the JSON transformer.
- Reuse of WSBind files that are created for SOAP web services applications in a JSON pipeline is not supported. WSBind files for use with JSON service provider applications must be generated by the JSON assistant.
- If a JSON payload is missing some of its required content when CICS transforms it, the equivalent fields within the data structures are not initialized when passed to the application program.
- CICS cannot transform integer values greater than the maximum value for a signed long (2^63 - 1) unless they are enclosed within quotes.
- Use of simple data types or arrays is not supported at the root of a JSON Schema. The JSON Schema is required to describe a JSON Object, though the JSON Object can be composed of simple data types and arrays.
- If an array is declared in a JSON schema with a maxItems value of 1, CICS serializes the array as a simple string or integer when generating JSON at runtime.

Important: The only supported characters for JSON property names are: `A-Z a-z _ :` for the first character and `A-Z a-z 0-9 _ : . -` for all subsequent characters.

The Axis2 web services support today has a number of options for development and deployment of applications and customizations. The following options are not supported:

- User-supplied application handlers - you must use the CICS supplied application handler class `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler`.
- User-written Axis2 Java applications.
- The SOAPFAULT and WS-Addressing APIs cannot be used with the JSON pipeline.

Container restrictions

Note that some pipeline containers are not populated when processing a JSON request. For more information, see Containers used in the pipeline in Configuring.

Differences in RESTful web services

On INQUIRE PIPELINE:

- SOAPLEVEL returns NOTSOAP
- The MTOMNOXOPST, MTOMST, SENDMTOMST, SOAPRNUM, SOAPVNUM, XOPDIRECTST, and XOPSUPPORTST attributes are not used.

On INQUIRE WEBSERVICE:

- The ARCHIVEFILE, BINDING, VALIDATIONST, XOPDIRECTST, and XOPSUPPORTST attributes are not used.
- WSDLFILE returns the name of the JSON schema file that is associated with the WEBSERVICE.

On the WEBSERVICE resource:

- The ARCHIVEFILE and VALIDATION parameters are not used and their values are ignored.
- WSDLFILE is the name of the JSON schema file that is associated with the WEBSERVICE.

Chapter 9. Runtime processing for web services

To send a request to a web service provider or to receive a request from a web service requester, your application (or wrapper program) must interact correctly with the web services support in CICS. You can also control the processing that takes place in the pipeline to determine how the inbound and outbound requests are handled.

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.

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 SERVICE** command to invoke a web service. The request and response are mapped to a data structure in container DFHWS-DATA. This method of invoking a service is not supported for JSON.

Procedure

1. Create a channel and populate it with containers. At the minimum, container DFHWS-DATA must be present. DFHWS-DATA 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 the following command:

   ```
   EXEC CICS INVOKE SERVICE(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.

   For more information, see "Local optimization for web services" on page 321.
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 a 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 must decide if the application program should roll back any changes. If a SOAP fault 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.

Local optimization for web services

You can use the provider application name in the web service binding file associated with the WEBSERVICE resource to enable local optimization of the web service request.

Using the INVOKE SERVICE command, you can specify the URIMAP(urimap) or URI(uri) where the uri is the URI of the web service to be invoked. If a URIMAP is specified, CICS uses the client mode URIMAP indicated to resolves the URI. If these options are omitted, CICS uses the URI specified in the web service description (WSDL) from which the WEBSERVICE was generated.

If the WEBSERVICE indicated is deployed in a requester mode PIPELINE, CICS invokes the remote web service. This is the most typical scenario.

If the WEBSERVICE indicated is deployed in a provider mode PIPELINE, CICS invokes the service locally. If you use this optimization, the EXEC CICS INVOKE SERVICE command is optimized to an EXEC CICS LINK command. This results in significant performance benefits, but introduces the following limitations:

- The PIPELINE is not driven and therefore no handler programs are used.
- The PIPELINE control containers are not present on the Channel. Some containers are present, including the DFHWS-DATA, DFHWS-OPERATION and DFHWS-URI containers. Any containers that normally contain XML are not present; this includes the DFH-REQUEST, DFHWS-BODY and DFHWS-XMLNS containers.
- Both the provider and requester applications must share the same copybooks and be implemented in the same programming language.
- Both the provider and requester applications share a single unit of work.
- If the web service is not expected to return a response, the EXEC CICS INVOKE SERVICE command does not return control to the application until after the target PROGRAM has ended.

If you want to use locally optimized web services but require data to be processed through a PIPELINE, use the `cics` URI format described here: "Options for controlling requester pipeline processing" on page 325. This mechanism is less efficient than using the fully optimized approach, but it avoids the processing cost of going out to the network.

Runtime limitations for code generated by the web services assistant

At runtime, 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 WebSphere 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 application data is encoded in EBCDIC or UTF-16. 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. See Support for UTF-16 in application data for more information on UTF-16 support.

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 SERVICE** command, then any containers it passes to CICS could potentially be reused and the data within them replaced. If you want to keep the data in these containers, create a new channel and copy the containers to it before you run the program. 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 SERVICE** command, rather than using the default channel that it was originally attached to. If your application program is using the **INVOKE SERVICE** 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 runtime, 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 runtime.

- 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 trailing white space characters are removed when generating SOAP messages. At runtime, inbound SOAP messages are parsed according to the XML Schema specification and all leading, trailing, and embedded white space is removed.

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 runtime, 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 SERVICE`** 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 runtime, then it is replaced with the equivalent entity. The XML entities that are supported are:

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.

Attention: CICS generates XML and JSON data from structured application data. If that application data contains bit patterns that look like preformatted XML, JSON, HTML, JPEG images, or any other meaningful content type, CICS is unaware of this semantic meaning, and processes such data as ordinary text or binary data. CICS does not attempt to recognize patterns in the data, or to process encoded data differently. For example, if the data contains pre-formatted XML (such as CDATA encoded text), that data is processed in the same way as any other data. Consider the following application data: `"An example: <here>"`. This example application-supplied data contains what looks like an XML tag, but it will be processed as raw text, resulting in the following XML representation: `"An example: < here > "`. If an application needs to generate XML itself, consider using `xsd:any` constructs in your XML schemas, or using XML-ONLY=TRUE in the Assistants.

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.

Customizing pipeline processing

In addition to providing your own message handlers, you can also use a set of global user exit points (GLUEs) to customize the processing that occurs for inbound and outbound web services in the pipeline.

Before you begin

You must understand the best practices for writing global user exit programs before customizing the pipeline. If you are customizing a service provider pipeline, you must be using the supplied DFHPITP or Axis2 application handler in your pipeline.

About this task

You can use the pipeline domain exits to access containers on a web services provider pipeline, a web services requester pipeline, or a web services requester pipeline that contains a security handler. The pipeline global user exits are described in detail in the *CICS Customization Guide*.

Procedure

1. Select which global user exit points to use:
   - Use XWSPRRWI, XWSPRROI, XWSPRROO, or XWSPRRWO GLUEs to access containers in a web services provider pipeline.

- Use XWSRQRWO, XWSRQROO, XWSROROI, or XWSRQRWI GLUEs to access containers in a web services requester pipeline.
- Use XWSSRRWO, XWSSRROO, XWSSRROI, or XWSSRRWI GLUEs to access containers in a secured web services requester pipeline.

2. Use the DFH$PIEX sample exit program to write your own global user exit program. DFH$PIEX is in the SDFHSAMP library. You are recommended to make the program threadsafe.
3. Enable the global user exit program.
4. Test your global user exit program to ensure it works correctly.

Related information:

Pipeline domain exits

The pipeline sample exit program

Writing global user exit programs

Defining, enabling, and disabling an exit program

Options for controlling requester pipeline processing

In service requester pipelines, message handlers can determine where the web service request is sent by changing the URI. CICS provides support for different URI formats so that you have much more flexibility in the way that the pipeline processes web service requests.

When the service requester pipeline reaches the end of its processing, you have the following options:

Linking to a program

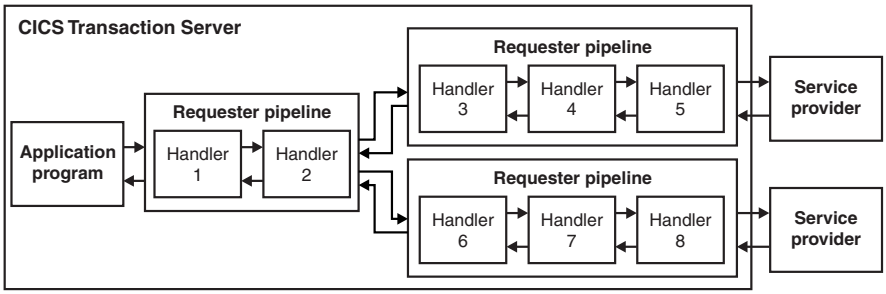
If you change the URI to the format `cics://PROGRAM/`*`program`*, where *program* is the name of the target application program, CICS passes the current channel and its containers or COMMAREA to the program using an **`EXEC CICS LINK`** command.

This processing is similar to the local optimization that occurs when the service requester and service provider applications are in the same CICS region. However, using this URI format provides the benefit of running the request through the pipeline and any custom message handlers first. The target application program must be able to handle the contents of the containers or COMMAREA.

Starting another requester mode pipeline

If you change the URI to the format `cics://PIPELINE/`*`pipeline`*`?targetServiceUri=`*`targetServiceUri`*, where *pipeline* is the name of a PIPELINE resource and *targetServiceUri* is the URI that you want to put in the DFHWS-URI container, CICS passes the current channel and its containers to the specified requester pipeline. Use this URI when you want to link two or more requester pipelines together before sending the request to the service provider. The number of requester pipelines that you can chain together is not limited.

In the following example, one generic requester pipeline supports one application. Message handlers 1 or 2 can change the URI for each request depending on the application data in the containers, sending the request to one of two requester pipelines that contain different message handlers.



Although the example shows only one service requester application, many applications could use the same generic requester pipeline and have their requests sent to different requester pipelines before the request is finally sent to the appropriate web service provider.

Sending the request straight to the provider mode pipeline

If you change the URI to the format `cics://SERVICE/`*`service`*`?targetServiceUri=`*`targetServiceUri`*, where *service* is the name of the target service and *targetServiceUri* is the path to the service, CICS resolves the request by matching the path to a URIMAP and passes the request to the correct provider pipeline. Use this option when you want to take advantage of processing the request through both the requester and provider pipelines without using the network.

This URI might also be useful where the requester and provider applications are written in different languages, or use different mapping levels, and expect different binary data.

Related tasks:

"Controlling requester pipeline processing using a URI" on page 327
In service requester pipelines, a message handler can determine where to send the web service request by changing the URI. By changing the URI format, you can choose to perform certain optimizations, such as starting another requester pipeline or starting a service provider pipeline without sending the request over the network.

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

Controlling requester pipeline processing using a URI

In service requester pipelines, a message handler can determine where to send the web service request by changing the URI. By changing the URI format, you can choose to perform certain optimizations, such as starting another requester pipeline or starting a service provider pipeline without sending the request over the network.

Before you begin

Decide which options you want to implement in your requester pipeline. See "Options for controlling requester pipeline processing" on page 325 for details.

About this task

The web service requester application can populate the DFHWS-URI container using the **EXEC CICS INVOKE SERVICE** command or, if no value is supplied by the application, CICS populates the container using the value in the web service binding file. To modify the URI, you must write a message handler that changes the contents of this container.

Procedure

1. Write a message handler to modify the DFHWS-URI container according to one of the following URI formats:
   - To link to an application program, use the URI `cics://PROGRAM/`*`program`*, where *program* is the target application program. No data transformation takes place, so you must ensure that the application program can process the contents of the containers on the current channel. The application program can pass either the current channel and containers or a COMMAREA.
   - To start a provider pipeline without going through the network, use the URI `cics://SERVICE/`*`service`*`?targetServiceUri=`*`targetServiceUri`*, where *service* is the name of the service and *targetServiceUri* is the path of the service. The transport handler uses the path of the service to locate the URIMAP resource that resolves the request and passes it to the correct provider pipeline. CICS does not use the name of the service in its processing.

     An error occurs if no URIMAP resource is installed for the service. The URIMAP resource definition must also specify USAGE(PIPELINE). The transport handler puts the value of the **targetServiceUri** parameter in the DFHWS-URI container and starts the provider pipeline.
   - To start another requester pipeline, use the URI `cics://PIPELINE/`*`pipeline`*`?targetServiceUri=`*`targetServiceUri`*, where *pipeline* is the name of the PIPELINE resource that you want to start and *targetServiceUri* is the value that you want to pass to the next pipeline in the DFHWS-URI container.

   Each type of URI has additional parameters that you can add as a query string. For more information about the format of these URIs and the rules for coding them, see the "DFHWS-URI container" on page 304.
2. Use an XML editor to add the message handler to the pipeline configuration file:

```
<service>
  <service_handler_list>
    <handler>
      <program>MYPROG</program>
    </handler>
  </service_handler_list>
</service>
```

3. Disable, discard, and reinstall the PIPELINE resource for the requester pipeline to include your new message handler program in the pipeline.
4. Install the message handler program in the CICS region.

Results

The next service request to run through the requester pipeline is processed by your new message handler.

What to do next

Test out the changes to your requester pipeline to make sure that the service requests are going to the correct location and that your message handler program is behaving as designed.

Related concepts:

"Options for controlling requester pipeline processing" on page 325
In service requester pipelines, message handlers can determine where the web service request is sent by changing the URI. CICS provides support for different URI formats so that you have much more flexibility in the way that the pipeline processes web service requests.

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

JSON web service restrictions

Use this reference material to understand capabilities that are not supported by JSON web services.

The following capabilities are not supported:

- Requester mode pipelines with JSON are not supported.
- Runtime validation of JSON data against schema is not supported. The value of the VALIDATION attribute of a WEBSERVICE resource that is used with a JSON payload is ignored.
- Use of namespaces in JSON data (Badgerfish or Mapped conventions) is not supported.
- JSON payloads sent to CICS must be encoded in UTF-8. No other encoding is supported. Similarly, JSON sent by CICS is always encoded in UTF-8.
- WebSphere MQ transports with JSON pipelines are not supported.
- Vendor transformer programs are not supported for use with the JSON transformer.
- Reuse of WSBind files that are created for SOAP web services applications in a JSON pipeline is not supported. WSBind files for use with JSON service provider applications must be generated by the JSON assistant.
- If a JSON payload is missing some of its required content when CICS transforms it, the equivalent fields within the data structures are not initialized when passed to the application program.
- CICS cannot transform integer values greater than the maximum value for a signed long (2^63 - 1) unless they are enclosed within quotes.
- Use of simple data types or arrays is not supported at the root of a JSON Schema. The JSON Schema is required to describe a JSON Object, though the JSON Object can be composed of simple data types and arrays.
- If an array is declared in a JSON schema with a maxItems value of 1, CICS serializes the array as a simple string or integer when generating JSON at runtime.

Important: The only supported characters for JSON property names are: `A-Z a-z _ :` for the first character and `A-Z a-z 0-9 _ : . -` for all subsequent characters.

The Axis2 web services support today has a number of options for development and deployment of applications and customizations. The following options are not supported:

- User-supplied application handlers - you must use the CICS supplied application handler class `com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler`.
- User-written Axis2 Java applications.
- The SOAPFAULT and WS-Addressing APIs cannot be used with the JSON pipeline.

Container restrictions

Note that some pipeline containers are not populated when processing a JSON request. For more information, see Containers used in the pipeline in Configuring.

Differences in RESTful web services

On INQUIRE PIPELINE:

- SOAPLEVEL returns NOTSOAP
- The MTOMNOXOPST, MTOMST, SENDMTOMST, SOAPRNUM, SOAPVNUM, XOPDIRECTST, and XOPSUPPORTST attributes are not used.

On INQUIRE WEBSERVICE:

- The ARCHIVEFILE, BINDING, VALIDATIONST, XOPDIRECTST, and XOPSUPPORTST attributes are not used.
- WSDLFILE returns the name of the JSON schema file that is associated with the WEBSERVICE.

On the WEBSERVICE resource:

- The ARCHIVEFILE and VALIDATION parameters are not used and their values are ignored.
- WSDLFILE is the name of the JSON schema file that is associated with the WEBSERVICE.

Chapter 10. 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 OASIS.

Note: CICS supports the November 2004 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.

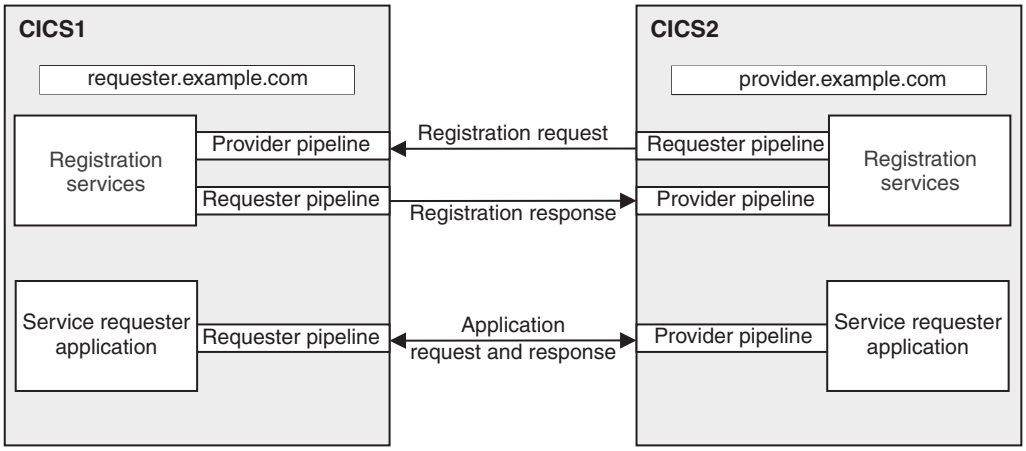


Figure 34. Registration services

Figure 29 on page 330 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 30 on page 331.

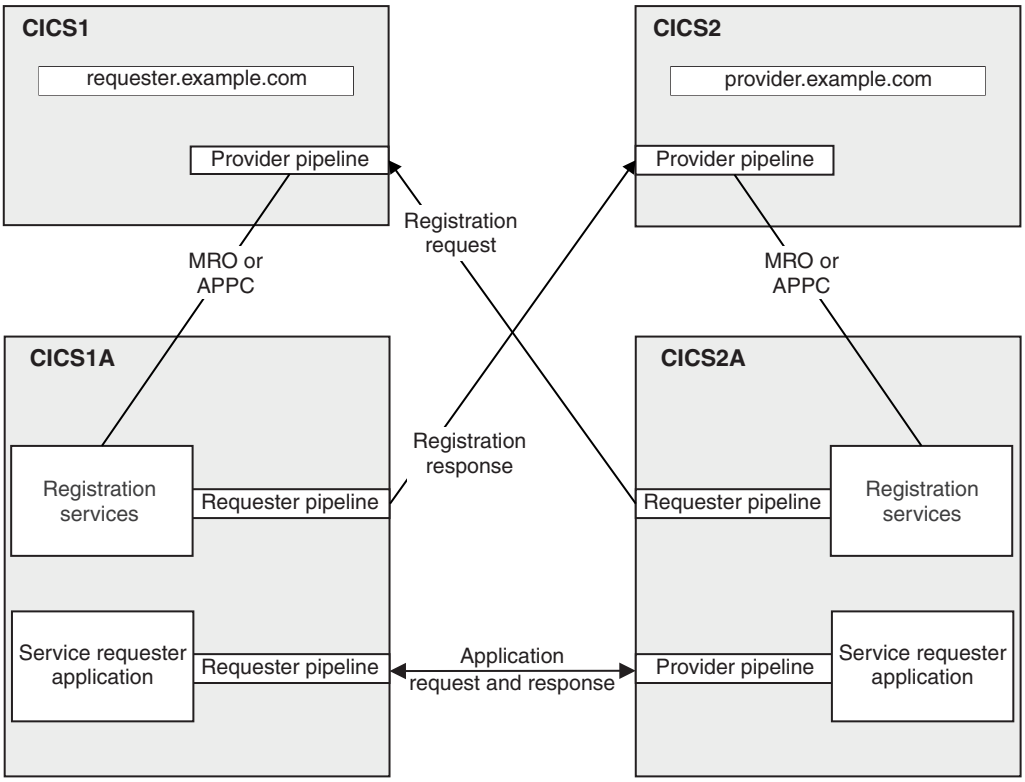


Figure 35. Registration services in a CICSPlex

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 30 on page 331 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 begin

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/cicsts53/pipeline/configs` directory, but the default file path might have been changed during CICS installation.

About this task

CICS support for web service transactions uses a CICS-supplied registration service. This registration service consists of a service provider and a service requester. You must install resources for both the service provider and the service requester; even if your applications are all service providers or all service requesters.

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 DFHWSAT 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:

Procedure

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. For more information about these groups, see Defining the WS-AT data set.

   If you want to share the DFHPIDIR file across CICS regions, the regions must be logically connected over MRO. You must install one data set per group of regions that are acting as a logical server.

   Tip: You are recommended not to share data sets between regions that are not logically connected.

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 registration service's provider PIPELINE resource. The PIPELINE is named DFHWSATP, and specifies the pipeline configuration file `/usr/lpp/cicsts/cicsts53/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 registration service's requester PIPELINE resource. The PIPELINE is named DFHWSATR, and specifies the pipeline configuration file `/usr/lpp/cicsts/cicsts53/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.

Results

CICS is now configured so that your service provider and requester applications can participate in distributed transactions using WS-AtomicTransaction and WS-Coordination protocols.

What to do next

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> element and a <service_parameter_list> element.

Before you begin

If you want your service provider application to 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.

About this task

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/cicsts53/samples/pipelines/wsatprovider.xml directory (where /usr/lpp/cicsts/cicsts53 is the default install directory for CICS files on z/OS UNIX).

Procedure

1. In the definition of your terminal handler, code a <headerprogram> element in the <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java>, or <cics_soap_1.2_handler_java> 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/wscoor</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>
   ```

   address is the IP address of the CICS region where the registration service provider pipeline is located.

   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>
http://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> element and a <service_parameter_list> element.

Before you begin

If you want your service requester application to 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.

About this task

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/cicsts53/samples/pipelines/wsatrequester.xml directory (where /usr/lpp/cicsts/cicsts53 is the default install directory for CICS files on z/OS UNIX).

Procedure

1. Code a <headerprogram> element in the <cics_soap_1.1_handler>, <cics_soap_1.2_handler>, <cics_soap_1.1_handler_java>, or <cics_soap_1.2_handler_java> element. Code the <program_name>, <namespace>, <localname>, and <mandatory> elements exactly as shown in the following example:

   ```
   <cics_soap_1.1_handler>
     <headerprogram>
       <program_name>DFHWSATH</program_name>
       <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoor</namespace>
       <localname>CoordinationContext</localname>
       <mandatory>true</mandatory>
     </headerprogram>
   </cics_soap_1.1_handler>
   ```

   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>
   ```

   address is the IP address of the CICS region where the registration service provider pipeline is located.

   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 of 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.

About this task

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:

Procedure

- 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 <cicswsa: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.

Example

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">
       <cicswsa:Address xmlns:cicswsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          http://clientIPaddress:clientPort/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
       </cicswsa:Address>
       <cicswsa:ReferenceProperties
          xmlns:cicswsa="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>
        </cicswsa: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 coordinator's 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.

About this task

If you need to access this information, you can either:

Procedure

- Look inside the contents of the `<cicswsa: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 `<cicswsa: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.

Example

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 11. Support for MTOM/XOP optimization of binary data

In standard SOAP messages, binary objects are base64-encoded and included in the message body, which increases their size by 33%. For very large binary objects, this size increase can significantly impact transmission time. Implementing MTOM/XOP provides a solution to this problem.

The SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specifications, often referred to as MTOM/XOP, define a method for optimizing the transmission of large base64Binary data objects within SOAP messages.

- The MTOM specification conceptually defines a method for optimizing SOAP messages by separating out binary data, that would otherwise be base64 encoded, and sending it in separate binary attachments using a MIME Multipart/Related message. This type of MIME message is called an *MTOM message*. Sending the data in binary format significantly reduces its size, thus optimizing the transmission of the SOAP message.
- The XOP specification defines an implementation for optimizing XML messages using binary attachments in a packaging format that includes but is not limited to MIME messages.

CICS implements support for these specifications in both requester and provider pipelines when the transport protocol is WebSphere MQ, HTTP, or HTTPS. As an alternative to including the base64Binary data directly in the SOAP message, CICS applications that are deployed as web service providers or requesters can use this support to send and receive MTOM messages with binary attachments.

You can configure this support by using additional options in the pipeline configuration file.

MTOM/XOP and SOAP

When MTOM/XOP is used to optimize a SOAP message, it is serialized into a MIME Multipart/Related message using XOP processing. The base64Binary data is extracted from the SOAP message and packaged as separate binary attachments within the MIME message, in a similar manner to e-mail attachments.

The size of the base64Binary data is significantly reduced because the attachments are encoded in binary format. The XML in the SOAP message is then converted to XOP format by replacing the base64Binary data with a special `<xop:Include>` element that references the relevant MIME attachment using a URI.

The modified SOAP message is called the *XOP document*, and forms the root document within the message. The XOP document and binary attachments together form the *XOP package*. When applied to the SOAP MTOM specification, the XOP package is a MIME message in MTOM format.

The root document is identified by referencing its Content-ID in the overall content-type header of the MIME message. Here is an example of a content-type header:

```
Content-Type: Multipart/Related; boundary=MIME_boundary;
 type="application/soap+xml"; start="<claim@insurance.com>"
```

The **start** parameter contains the Content-ID of the XOP document. If this parameter is not included in the content-type header, the first part in the MIME message is assumed to be the XOP document.

The order of the attachments in the MIME message is unimportant. In some messages for example, the binary attachments could appear before the XOP document. An application that handles MIME messages must not rely on the attachments appearing in a specific order. For detailed information, read the MTOM/XOP specifications.

The following example demonstrates how a simple SOAP message that contains a JPEG image is optimized using XOP processing. The SOAP message is as follows:

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xmime="http://www.w3.org/2003/12/xop/mime">
 <soap:Body>
 <submitClaim>
  <accountNumber>5XJ45-3B2</accountNumber>
  <eventType>accident</eventType>
  <image xmime:contentType="image/jpeg" xsi:type="base64binary">4f3e..(encoded image)</image>
 </submitClaim>
 </soap:Body>
</soap:Envelope>
```

An MTOM/XOP version of this SOAP message is as follows:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
 type="application/soap+xml"; start="<claim@insurance.com>" 1

--MIME_boundary
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim@insurance.com> 2

<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xop='http://www.w3.org/2004/08/xop/include'
 xmlns:xop-mime='http://www.w3.org/2005/05/xmlmime'>
 <soap:Body>
 <submitClaim>
  <accountNumber>5XJ45-3B2</accountNumber>
  <eventType>accident</eventType>
  <image xop-mime:content-type='image/jpeg'><xop:Include href="cid:image@insurance.com"/></image> 3
 </submitClaim>
 </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <image@insurance.com> 4

...binary JPG image...

--MIME_boundary--
```

1. The **start** parameter indicates which part of the MIME message is the root XOP document.
2. The Content-ID value identifies a part of the MIME message. In this case it is the root XOP document.
3. The <xop:Include> element references the JPEG binary attachment.
4. The Content-ID identifies the JPEG in the binary attachment.

MTOM messages and binary attachments in CICS

CICS supports and controls the handling of MTOM messages in both web service provider and requester pipelines using an MTOM handler program and XOP processing.

You configure and enable the MTOM support using the pipeline configuration file. If MTOM support is enabled for a pipeline, CICS unpacks inbound MTOM messages automatically and packages outbound messages. If MTOM support is not enabled for a pipeline and CICS receives an MTOM message, Java-based pipelines accept the inbound MTOM message, however other SOAP pipelines reject the inbound MTOM message with a SOAP fault.

Configuration options for Java-based pipelines

You can configure a provider pipeline to perform the following tasks:

- Accept MTOM messages, but never send MTOM response messages.
- Accept MTOM messages and always send MTOM response messages.
- Process XOP documents and binary attachments in Axis2 mode.

You can configure a requester pipeline to perform the following tasks:

- Never send an MTOM message, but accept MTOM response messages.
- Always send MTOM messages and accept MTOM response messages.
- Process XOP documents and binary attachments in Axis2 mode.

Configuration options for pipelines that do not support Java

You can configure a provider pipeline to perform the following tasks:

- Accept MTOM messages, but never send MTOM response messages.
- Accept MTOM messages and send the same type of response message.
- Accept MTOM messages, but only send MTOM messages when there are binary attachments present.
- Accept MTOM messages and always send MTOM response messages.
- Process XOP documents and binary attachments in direct or compatibility mode.

You can configure a requester pipeline to perform the following tasks:

- Never send an MTOM message, but accept MTOM response messages.
- Only send MTOM messages when there are binary attachments, and accept MTOM response messages.
- Always send MTOM messages and accept MTOM response messages.
- Process XOP documents and binary attachments in direct or compatibility mode.

Modes of support

There are three modes of support provided in the pipeline to handle XOP documents and any associated binary attachments.

Axis2 mode
: Axis2 mode is used when the terminal handler of your web services pipeline is either the <cics_soap_1.1_handler_java> or the <cics_soap_1.2_handler_java> message handler.

Direct mode

In direct mode, the binary attachments associated with an inbound or outbound MTOM message are passed in containers through the pipeline and handled directly by the application, without the need to perform any data conversion.

Compatibility mode
Compatibility mode is used when the pipeline processing requires the message to be in standard XML format, with any binary data stored as base64Binary fields within the message. For inbound messages, the XOP document and binary attachments are reconstituted into a standard XML message, either at the beginning of the pipeline when Web Services Security is enabled, or at the end of the pipeline when web service validation is enabled. For outbound messages, a standard XML message is created and passed along the pipeline. This XML message is converted to XOP format by the MTOM handler just before CICS sends it.

Compatibility mode is much less efficient than direct mode because binary data gets converted to base64 format and back again. However, it does allow your web services to interoperate with other MTOM/XOP web service requesters and providers without needing to change your applications.

Inbound MTOM message processing for pipelines that do not support Java

When the MTOM handler is enabled in pipelines that do not support Java, it checks the headers of the inbound message in the DFHREQUEST or DFHRESPONSE container to determine the format of the message during the transport handling processing.

When a MIME Multipart/Related message is received, the MTOM handler unpackages the message as follows:

1. It puts the headers and binary data from each binary attachment into separate containers.
2. It puts the list of containers in the DFHWS-XOP-IN container.
3. It puts the XOP document, which formed the root of the message, back in the DFHREQUEST or DFHRESPONSE container, replacing the original message.

If there are no binary attachments, the XOP document is handled as a normal XML message and no XOP processing is required. If there are any binary attachments, XOP processing is enabled for the message.

If XOP processing is enabled, the MTOM handler checks the pipeline properties to determine if the current message should be processed in direct or compatibility mode, and puts this information in the DFHWS-MTOM-IN container.

In provider mode, the MTOM handler also creates the DFHWS-MTOM-OUT container to determine how the outbound response message should be processed.

Direct mode

When you are using CICS web services support, that is, when a service provider pipeline uses the DFHPITP application handler or a service requester pipeline is invoked using **INVOKE WEBSERVICE**, the pipeline can process the XOP document and binary attachments in direct mode.

In this mode, the XOP document and associated containers are passed by the MTOM handler to the next message handler in the pipeline for processing. The CICS web services support interprets the <xop:Include> elements. If the base64Binary field is represented as a container in the application data structure, then the attachment container name is stored in the structure. If the field is represented as a variable or fixed length string, the contents of the container are copied to the relevant application data structure field. The data structure is then passed to the application program.

Compatibility mode

If your pipeline is configured to use a custom application handler, or Web Services Security is also enabled, the message is processed in compatibility mode. In this mode, the XOP document and binary attachments are immediately reconstituted into a SOAP message using XOP processing, so that the content can be successfully processed in the pipeline. The XOP processing performs the following tasks:

1. Scans the XOP document for <xop:Include> elements, replacing each occurrence with the binary data from the referenced attachment in base64-encoded format.
2. Discards the DFHWS-XOP-IN container and all of the attachment containers.

The reconstituted SOAP message is then passed to the next handler in the pipeline to be processed as normal.

If web service validation is enabled, the pipeline switches to compatibility mode when the message reaches the application handler. The message is reconstituted into a SOAP message, validated, and passed to the application.

Outbound MTOM message processing for pipelines that do not support Java

When a pipeline that does not support Java is configured to send outbound MTOM messages, the web service and pipeline properties are checked to determine how the message should be processed and sent.

These properties are stored in two containers, DFHWS-MTOM-OUT and DFHWS-XOP-OUT. In a requester mode pipeline, these containers are created by CICS when the application issues the **EXEC CICS INVOKE WEBSERVICE** command. In a provider mode pipeline, the DFHWS-MTOM-OUT container is already initialized with the options that were determined when the inbound message was received.

If the outbound message can be processed in direct mode, the optimization of the message takes place immediately. If the outbound message has to be processed in compatibility mode, the optimization takes place at the very end of the pipeline processing.

If you have not deployed your web service provider or requester application using the CICS web services assistant, or if you have web service validation enabled or Web Services Security enabled in your pipeline, the outbound message is processed in compatibility mode.

Direct mode

In direct mode, the following processing takes place:

1. An XOP document is constructed from the application's data structure in container DFHWS-DATA. Any binary fields that are equal to or larger in size than 1500 bytes are identified, and the binary data and MIME headers describing the binary attachment are put in separate containers. If the binary data is already in a container, that container is used directly as the attachment. A `<xop:Include>` element is then inserted in the XML in place of the usual base64-encoded binary data using a generated Content-ID. For example:

   ```
   <xop:Include href="cid:generated-content-ID-value"
    xmlns:xop="http://www.w3.org/2004/08/xop/include">
   ```

2. All of the containers are added to the attachment list in the DFHWS-XOP-OUT container.
3. When the SOAP handler has processed DFHWS-DATA, the XOP document and SOAP envelope are stored in the DFHREQUEST or DFHRESPONSE container and processed through the pipeline.
4. When the last message handler has finished, the MTOM handler packages the XOP document and binary attachments into a MIME Multipart/Related message and sends it to the web service requester or provider. The DFHWS-XOP-OUT container and any associated containers are then discarded.

Compatibility mode

If the pipeline is not capable of handling the XOP document directly, then the following processing takes place:

1. The SOAP body is constructed in DFHWS-DATA from the application data structure and processed in the pipeline as normal.
2. When the final handler has finished processing the message, the MTOM handler checks the options in the DFHWS-MTOM-OUT container to determine whether MTOM should be used, optionally taking into account whether any binary attachments are present. If the MTOM handler determines that MTOM is not required, no XOP processing takes place and a SOAP message is sent by CICS as normal.
3. If the MTOM handler determines that the outbound message should be sent in MTOM format, the XOP processing scans the message for eligible fields to split the data out into binary attachments. For a field to be eligible, it must have the MIME **`contentType`** attribute specified on the element and the associated binary value must consist of valid base64Binary data in canonical form. The size of the data must be greater than 1500 bytes. The XOP processing creates the binary attachments and attachment list, and then replaces the fields with `<xop:Include>` elements.
4. The MTOM handler packages the XOP document and binary attachments as a MIME Multipart/Related message and CICS sends it to the web service requester or provider.

Restrictions when using MTOM/XOP

To support MTOM/XOP you can either specify the `<mtom>` element in your pipeline configuration file or enable the MTOM handler in your pipeline. However, there are restrictions associated with each method.

Restrictions for Java-based pipelines

Specifying the `<mtom>` element in the a pipeline configuration file enables MTOM/XOP support for your Java-based pipeline. However, there are restrictions with this MTOM/XOP implementation.

DFHPITP application handler
The Axis2 mode of MTOM/XOP support cannot be used with pipelines that specify DFHPITP as the application handler.

WS-Security
The Axis2 mode of MTOM/XOP support cannot be used with pipelines that use WS-Security configurations that require XML signatures.

Using the INQUIRE PIPELINE command
If an **INQUIRE PIPELINE** command is issued against a Java-based pipeline using the Axis2 mode of MTOM/XOP support, the **Mtomst**, **Sendmtomst**, **Mtomnoxopst**, **Xopsupportst**, and **Xopdirectst** attributes report as Nomtom. For more information, see INQUIRE PIPELINE.

Restrictions for other SOAP pipelines

Enabling the MTOM handler in the pipeline means that you can support web service implementations that use the MTOM/XOP optimization. The compatibility mode option means that you can interoperate with these web services without needing to change your web service applications. However, there are certain situations where you cannot use MTOM/XOP or its use is restricted.

Using the CICS web services assistant
The direct mode optimization for MTOM/XOP is only available if you are using DFHWS2LS at a mapping level of at least 1.2, and the WSDL document contains at least one field of type xsd:base64Binary. Web services that are enabled using DFHLS2WS are not eligible for XOP optimization.

Web services generated using DFHLS2WS with CHAR-VARYING=BINARY specified may be eligible for the MTOM/XOP optimizations. Other web services generated using DFHLS2WS do not contain binary data and are not eligible for the MTOM/XOP optimizations, but will work normally in a PIPELINE that supports MTOM/XOP.

Provider pipelines
CICS provides a default application handler called DFHPITP that can be configured in a provider pipeline. This application handler is capable of handling XOP documents and creating the necessary containers to support the pipeline processing in both direct and compatibility mode. If you are using your own application handler in a provider pipeline, and want to enable MTOM/XOP, you should configure the pipeline to run in compatibility mode.

Requester pipelines
If your applications use the **INVOKE WEBSERVICE** command, CICS handles the optimization of the SOAP message for you in direct and compatibility mode. If you are using the program DFHPIRT to start the pipeline, you can only send and receive MIME Multipart/Related messages in compatibility mode.

Web Services Security
If you enable the MTOM handler in the pipeline configuration file to run in direct mode, and you also enable the Web Services Security message handler, the pipeline only supports the handling of MTOM messages in compatibility mode.

Handling binary data
When you have large binary data to include in your web service, for example a graphic file such as a JPEG, you can use MTOM/XOP to optimize the size of the message that is sent to the service provider or

requester. The minimum size of binary data that can be optimized using MTOM/XOP is 1500 bytes. If the binary data in a field is less than 1500 bytes, CICS does not optimize the field.

As stated in the XOP specification, there should be no white space in the base64Binary data. Any application programs that produce base64Binary data must use the canonical form. If the base64Binary data in an outbound message does contain white space, CICS does not convert the data to a binary attachment. When base64Binary data is generated by CICS, the fields are provided in canonical form and therefore contain no white space.

The **contentType** attribute must be present on base64Binary fields for XOP processing to occur in compatibility mode on outbound messages. The **contentType** attribute must not be present on hexBinary fields.

Web service validation

If you turn on web service validation the following pipeline processing takes place:

- If an inbound XOP document has been passed through the pipeline in direct mode, CICS automatically switches to compatibility mode and converts it back to standard XML when CICS web service support is about to validate the document.
- An outbound SOAP message is generated as standard XML and is processed in compatibility mode.

The extra pipeline processing is required because the validation processing cannot handle the contents of XOP documents.

Configuring CICS to support MTOM/XOP

To support MTOM messages in CICS, you must specify the correct MTOM/XOP support for your type of pipeline in your pipeline configuration files.

Configuring MTOM/XOP support for Java-based pipelines

To configure MTOM/XOP support for Java-based pipelines, you must add the <mtom> element to your pipeline configuration files.

Before you begin

Before performing this task, you must identify or create the pipeline configuration files to which you will add configuration information for MTOM/XOP.

About this task

If the <mtom> element is defined in your pipeline configuration file, MTOM support is enabled for all inbound and outbound messages. However, if this element is not specified in the pipeline configuration file, then MTOM support is enabled for only inbound messages.

Procedure

Add a <mtom> element to your pipeline configuration file. This element should be defined after the optional <addressing> element and before the optional <headerprogram> element.

Example

For a provider or requester mode pipeline, you could specify:

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <addressing></addressing>
  <mtom></mtom>
  <headerprogram>
    <program_name>HDRPROG4</program_name>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler_java>
```

Configuring MTOM/XOP for other SOAP pipelines

To configure MTOM/XOP support for pipelines that do not use the <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> handlers, you must add the MTOM handler to your pipeline configuration files.

Before you begin

Before performing this task, you must identify or create the pipeline configuration files to which you will add configuration information for MTOM/XOP.

Procedure

1. Add a <cics_mtom_handler> element to your pipeline configuration file. This element should be first in the <provider_pipeline> element, and the last element before the <service_parameter_list> in the <requester_pipeline> element. Code the following elements:

   ```
   <cics_mtom_handler>
     <dfhmtom_configuration version="1">
     </dfhmtom_configuration>
   </cics_mtom_handler>
   ```

   The <dfhmtom_configuration> element is a container for the other elements in the configuration. If you want to accept the default settings for MTOM/XOP processing, you can specify an empty element as follows:

   ```
   <cics_mtom_handler/>
   ```

2. Optional: Code an <mtom_options> element. In both a service provider and service requester pipeline, this element specifies whether the outbound message should be packaged as an MTOM message.
   a. Code the **send_mtom** attribute to define if the outbound message should be sent as an MTOM message. For details of this attribute, see "The <mtom_options> element" on page 259.
   b. Code the **send_when_no_xop** attribute to define if the outbound message should be sent as an MTOM message when there are no binary attachments present. For details of this attribute, see "The <mtom_options> element" on page 259.
3. Optional: Code a <xop_options> element with an **apphandler_supports_xop** attribute. This specifies if the application handler is capable of handling XOP documents directly. If you do not include this attribute, the default depends on whether the <apphandler> element specifies DFHPITP or another program. For details of this attribute, see "The <xop_options> element" on page 260.
4. Optional: Code a <mime_options> element with a **content_id_domain** attribute. This specifies the domain name that should be used when generating MIME

content-ID values, that are used to identify binary attachments. For details of this attribute, see "The <mime_options> element" on page 261.

Example

The following example shows a completed <cics_mtom_handler> element in which all the optional elements are present:

```
<provider_pipeline>
  <cics_mtom_handler>
    <dfhmtom_configuration version="1">
      <mtom_options send_mtom="same" send_when_no_xop="no" />
      <xop_options apphandler_supports_xop="yes" />
      <mime_options content_id_domain="example.org" />
    </dfhmtom_configuration>
  </cics_mtom_handler>
....
</provider_pipeline>
```

Chapter 12. Support for Web Services Addressing

CICS supports services that use the Worldwide Web Consortium (W3C) Web Services Addressing (WS-Addressing) specifications. This family of specifications provides transport-independent mechanisms to address web services and facilitate end-to-end addressing.

CICS ensures that your existing web service applications can accept requests from web services that use WS-Addressing. You can also create new web services that use endpoint references and message addressing properties in SOAP messages.

WS-Addressing adds addressing information, in the form of Message Addressing Properties (MAPs), to SOAP message headers. MAPs include messaging information, such as a unique message ID and endpoint references that detail where the message came from, where the message is going to, and where reply or fault messages are to be sent. An endpoint reference (EPR) is a specific type of MAP, which includes the destination address of the message, optional reference parameters for use by the application, and optional metadata.

Features of the WS-Addressing support

CICS includes the following features to support WS-Addressing:

- Your web service requester and provider applications can interact with other services that are using WS-Addressing without requiring you to redeploy them. A new message handler, the addressing message handler DFHWSADH, in the pipeline routes messages that contain WS-Addressing information to the specified web service.
- You can write an application that uses the WS-Addressing API commands to create an endpoint reference and to create, update, delete, and query an addressing context.
- You can route response messages to endpoints other than the requester endpoint; for example, you can route fault messages to a dedicated fault handler.
- You can pass reference parameters to applications as part of the MAPs in the SOAP header.

Support for WS-Addressing specifications and interoperability

By default, CICS supports the recommendation specifications:

- W3C WS-Addressing 1.0 - Core
- W3C WS-Addressing 1.0 - SOAP Binding
- W3C WS-Addressing 1.0 - Metadata

These specifications are identified by the `http://www.w3.org/2005/08/addressing` namespace. Unless otherwise stated, WS-Addressing semantics that are described in this documentation refer to the recommendation specifications.

For interoperability, CICS also supports the submission specification:

- W3C WS-Addressing- Submission

This specification is identified by the `http://schemas.xmlsoap.org/ws/2004/08/addressing` namespace. Use the submission specification only if you must

interoperate with a client or web service provider that implements the submission specification.

Web Services Addressing overview

Web Services Addressing (WS-Addressing) provides a standard framework for specifying the endpoints of a SOAP message. This framework is transport-neutral and improves the interoperability of web services that use different transport mechanisms. The WS-Addressing specification introduces message addressing properties and endpoint references.

Web Services Addressing (WS-Addressing) is a Worldwide Web Consortium (W3C) specification that improves interoperability between web services by defining a standard way to address web services and provide addressing information in SOAP messages. SOAP messages can be sent over a variety of transport mechanisms, including HTTP and WebSphere MQ, each of which stores destination information for the message in a different way.

Existing CICS web services that are deployed in a pipeline configured to use WS-Addressing can use the default WS-Addressing settings without requiring any changes. To take full advantage of the WS-Addressing capabilities, use the WS-Addressing API commands. The WS-Addressing implementation supports one SOAP fault for each WSDL operation.

Message addressing properties

Message addressing properties (MAPs) are a set of well defined WS-Addressing properties that can be represented as elements in SOAP headers. MAPs provide a standard way of conveying information, such as the endpoint to which message replies must be directed, or information about the relationship that the message has with other messages. The MAPs that are defined by the WS-Addressing specification are summarized in the following table.

Table 17. Message addressing properties defined by the WS-Addressing specification

Abstract WS-Addressing MAP name	SOAP WS-Addressing MAP name	MAP content type	Multiplicity	Description
[action]	<wsa:Action>	xs:anyURI	1..1	An absolute URI that uniquely identifies the semantics of the message. This value is required.
[destination]	<wsa:To>	xs:anyURI in the SOAP message EndpointReference in the addressing context	0..1	The absolute URI that specifies the address of the intended receiver of the message. If this value is not specified, it defaults to the anonymous URI that is defined in the specification: `http://www.w3.org/2005/08/addressing/anonymous`. In the addressing context, the <wsa:To> MAP is represented as an EPR. When the <wsa:To> is sent as part of a SOAP message it is split into its address and its reference parameters, as defined by the schema.
[reference parameters] *	[reference parameters]*	xs:any	0..unbounded	Parameters that correspond to <wsa:ReferenceParameters> properties of the endpoint reference to which the message is addressed. This value is optional.

Table 17. Message addressing properties defined by the WS-Addressing specification (continued)

Abstract WS-Addressing MAP name	SOAP WS-Addressing MAP name	MAP content type	Multiplicity	Description
[source endpoint]	<wsa:From>	EndpointReference	0..1	A reference to the endpoint from which the message originated. This value is optional.
[reply endpoint]	<wsa:ReplyTo>	EndpointReference	0..1	An endpoint reference for the intended receiver of replies to this message. This value is optional. If this value is not specified, it defaults to `http://www.w3.org/2005/08/addressing/anonymous`.
[fault endpoint]	<wsa:FaultTo>	EndpointReference	0..1	An endpoint reference for the intended receiver of faults relating to this message. This value is optional and defaults to the value of the <wsa:ReplyTo> MAP.
[relationship] *	<wsa:RelatesTo>	xs:anyURI plus optional attribute of type xs:anyURI	0..unbounded	A pair of values that indicate how this message relates to another message. The contents of this element conveys the <wsa:MessageID> of the related message. An optional attribute conveys the relationship type. This value is optional. If this value is not specified, it defaults to `http://www.w3.org/2005/08/addressing/reply`.
[message id]	<wsa:MessageID>	xs:anyURI		An absolute URI that uniquely identifies the message. This value is optional; if not supplied, CICS generates a value for outbound requests and responses.

The following example of a SOAP message contains WS-Addressing MAPs:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://w3.org/2005/08/addressing"
            xmlns:example="http://example.ibm.com/namespace">
  <S:Header>
      ...
      <wsa:To>http://example.ibm.com/enquiry</wsa:To>
      <wsa:ReplyTo>
        <wsa:Address>http://example.ibm.com/enquiryReply</wsa:Address>
      </wsa:ReplyTo>
      <wsa:Action>...</wsa:Action>
      <example:AccountCode wsa:IsReferenceParameter='true'>123456789</example:AccountCode>
      <example:DiscountId wsa:IsReferenceParameter='true'>ABCDEFG</example:DiscountId>
      ...
  </S:Header>
  <S:Body>
  ...
  </S:Body>
</S:Envelope>
```

Endpoint references

An endpoint reference is a specific type of MAP, which provides a standard mechanism to encapsulate information about specific endpoints. Endpoint references can be sent to other parties and used to target the web service endpoint that they represent. The following table summarizes the information model for endpoint references.

Table 18. Information model for endpoint references

Abstract property name	Property type	Multiplicity	Description
[address]	xs:anyURI	1..1	The absolute URI that specifies the address of the endpoint.
[reference parameters] *	xs:any	0..unbounded	Namespace qualified element information items that are required to interface with the endpoint.
[metadata]	xs:any	0..unbounded	Description of the behavior, policies, and capabilities of the endpoint.

The following XML fragment illustrates an endpoint reference. The `<wsa:EndpointReference>` element references the endpoint at the URI `http://example.ibm.com/enquiry` and contains metadata specifying the interface to which the endpoint reference refers and some application-specific reference parameters.

```
<wsa:EndpointReference
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:example="http://example.ibm.com/namespace">
  <wsa:Address>http://example.ibm.com/enquiry</wsa:Address>
  <wsa:Metadata
      xmlns:wsdli="http://www.w3.org/ns/wsdl-instance"
      wsdli:wsdlLocation="http://example.ibm.com/wsdl/wsdl-location.wsdl">
    <wsam:InterfaceName>example:reservationInterface</wsam:InterfaceName>
  </wsa:Metadata>
  <wsa:ReferenceParameters>
      <example:AccountCode>123456789</example:AccountCode>
      <example:DiscountId>ABCDEFG</example:DiscountId>
   <wsa:ReferenceParameters>
</wsa:EndpointReference>
```

WS-Addressing MAPs of type `wsa:EndpointReferenceType` are: <wsa:From>, <wsa:ReplyTo>, and <wsa:FaultTo>. However, the <wsa:To> MAP is defined in the WS-Addressing 1.0 standard as having a type of `xs:anyURI`. For simplicity CICS treats <wsa:To> MAPs in the addressing context as EPRs. When a <wsa:To> MAP is sent as part of a SOAP message, CICS splits it into its address and reference parameters, as required by the standard.

Default namespaces

The following prefix and corresponding namespaces are referred to throughout the WS-Addressing documentation:

Table 19. Prefix and corresponding namespace

Default prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
wsa	http://www.w3.org/2005/08/addressing (Recommendation schema) http://schemas.xmlsoap.org/ws/2004/08/addressing (Submission schema)
wsam	http://www.w3.org/2007/05/addressing/metadata

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

Configuring a requester pipeline for Web Services Addressing

To configure a requester pipeline to support Web Services Addressing (WS-Addressing), you must add an addressing handler to your pipeline configuration file.

Before you begin

You must identify or create the pipeline configuration file to add the configuration information for WS-Addressing. You must also decide which of the WS-Addressing specifications to use. Use the *W3C WS-Addressing 1.0 Core* specification where possible.

About this task

You can add support for WS-Addressing in one of two ways:

- If the SOAP pipeline uses Java, the SOAP processing is handled by Axis2 and you can use the support provided by this technology to handle requests that use WS-Addressing. All of the header handling is handled by Axis2 and it is important that you do not add the DFHWSADH header processing program to the pipeline. You can use your own header processing programs. For better performance, write Axis2 handlers in Java if you want to process SOAP headers.
- If the SOAP pipeline does not use Java, you must add the CICS-supplied header processing program DFHWSADH to handle the requests.

Procedure

- If the SOAP pipeline uses a `<cics_soap_1.1_handler_java>` or `<cics_soap_1.2_handler_java>` element, add an `<addressing>` element to the pipeline configuration file. Include one `<namespace>` element that contains the specification that you want to use on the request message, which can be different to the response message; for example, you can always send a request that complies with the W3C core specification, even if the response message uses the submission specification. Axis2 supports both WS-Addressing specifications on inbound messages.

  The following example shows how you might configure the requester pipeline:

```
<requester_pipeline>
   <service>
      <service_handler_list>
         <cics_soap_1.1_handler_java>
            <jvmserver>JVMSERV1</jvmserver>
            <addressing>
               <namespace>http://www.w3.org/2005/08/addressing</namespace>
            </addressing>
         </cics_soap_1.1_handler_java>
      </service_handler_list>
   </service>
</requester_pipeline>
```

  The `<jvmserver>` element contains the name of the JVMSERVER resource that supports Axis2.

- If the SOAP pipeline does not use Java, add a CICS addressing header program in the <cics_soap_1.1_handler> or <cics_soap_1.2_handler> to the pipeline configuration file. The following example shows how you might configure the requester pipeline:

```
<requester_pipeline>
   <service>
      <service_handler_list>
         <cics_soap_1.1_handler>
            <headerprogram>
               <program_name>DFHWSADH</program_name>
               <namespace>http://www.w3.org/2005/08/addressing</namespace>
               <localname>*</localname>
               <mandatory>true</mandatory>
            </headerprogram>
         </cics_soap_1.1_handler>
      </service_handler_list>
   </service>
</requester_pipeline>
```

  Code the <program_name>, <localname>, and <mandatory> elements exactly as shown. Set <namespace> to http://www.w3.org/2005/08/addressing to use the *W3C WS-Addressing 1.0 Core* specification or http://schemas.xmlsoap.org/ws/2004/08/addressing to use the *W3C WS-Addressing Submission* specification.

  The order of header processing programs is not guaranteed. If you define other header processing programs, add them in a subsequent CICS SOAP handler element in your <service_handler_list> element. The DFHWSADH header handler must be in the first SOAP handler element.

Results

Your requester pipeline is now configured to support WS-Addressing.

What to do next

Create a PIPELINE resource that points to the configuration file. If you are using a Java-based SOAP pipeline, ensure that a JVMSERVER resource is enabled to handle the Axis2 processing.

Configuring a provider pipeline for Web Services Addressing

To configure a provider pipeline to support Web Services Addressing (WS-Addressing), you must add an addressing handler to your pipeline configuration file.

Before you begin

You must identify or create the pipeline configuration file to add the configuration information for WS-Addressing. You must also decide which of the WS-Addressing specifications to use. Use the *W3C WS-Addressing 1.0 Core* specification where possible.

About this task

You can add support for WS-Addressing in one of two ways:

- If the SOAP pipeline uses Java, the SOAP processing is handled by Axis2 and you can use the support provided by this technology to handle requests that use WS-Addressing. All of the header handling is handled by Axis2 and it is

important that you do not add the DFHWSADH header processing program to the pipeline. You can use your own header processing programs. For better performance, write Axis2 handlers in Java if you want to process SOAP headers.

- If the SOAP pipeline does not use Java, you must add the CICS-supplied header processing program DFHWSADH to handle the requests.

Procedure

- If the SOAP pipeline uses a <cics_soap_1.1_handler_java> or <cics_soap_1.2_handler_java> element, add an <addressing> element to the pipeline configuration file. You can optionally include one or more <namespace> elements. This element contains the specification that you want to use on the outbound message, which can be different to the inbound message; for example, you can always send an outbound response that complies with the W3C core specification, even if the inbound message uses the submission specification. If you exclude this element, Axis2 uses the same specification on the outbound message as the inbound message. Axis2 supports both WS-Addressing specifications on inbound messages.

  The following example shows how you might configure the provider pipeline:

```
<provider_pipeline>
   <terminal_handler>
      <cics_soap_1.1_handler_java>
          <jvmserver>JVMSERV1</jvmserver>
          <addressing>
             <namespace>http://www.w3.org/2005/08/addressing</namespace>
          </addressing>
      </cics_soap_1.1_handler_java>
    </terminal_handler>
</provider_pipeline>
```

  The <jvmserver> element contains the name of the JVMSERVER resource that supports Axis2.

- If the SOAP pipeline does not use Java, add the CICS addressing header program DFHWSADH to the SOAP handler in the pipeline configuration file. The following example shows how you might configure the provider pipeline:

```
<provider_pipeline>
   <terminal_handler>
      <cics_soap_1.1_handler>
         <headerprogram>
             <program_name>DFHWSADH</program_name>
             <namespace>http://www.w3.org/2005/08/addressing</namespace>
             <localname>*</localname>
             <mandatory>true</mandatory>
         </headerprogram>
      </cics_soap_1.1_handler>
   </terminal_handler>
</provider_pipeline>
```

  Code the <program_name>, <localname>, and <mandatory> elements exactly as shown. Set <namespace> to http://www.w3.org/2005/08/addressing to use the *W3C WS-Addressing 1.0 Core* specification or http://schemas.xmlsoap.org/ws/2004/08/addressing to use the *W3C WS-Addressing Submission* specification.

  The order of header processing programs is not guaranteed. If you define other header processing programs, add them in another CICS SOAP handler element in a <service_handler_list> element. The DFHWSADH header handler must be in the last SOAP handler element.

Results

Your provider pipeline is now configured to support WS-Addressing.

What to do next

Create a PIPELINE resource that points to the configuration file. If you are using a Java-based SOAP pipeline, ensure that a JVMSERVER resource is enabled to handle the Axis2 processing.

Creating a web service that uses WS-Addressing

To create a web service from a WSDL document that uses Web Services Addressing (WS-Addressing), use parameters on the web services assistant to handle the conversion from XML to language structures.

About this task

You can use the web services assistant job, DFHWS2LS, to control how an end point reference (EPR) is handled in the WSDL document and determine whether CICS constructs default input, output, and fault actions.

Procedure

1. Set the **MINIMUM-RUNTIME** parameter on the web services assistant, DFHWS2LS, to `3.0` or higher. A runtime level of at least 3.0 ensures that any generated web service binding fully supports WS-Addressing and can interoperate with other web services platforms.
2. Set the **MAPPING-LEVEL** parameter on the web services assistant, DFHWS2LS, to `3.0` or higher.
3. Set the **WSADDR-EPR-ANY** parameter to `TRUE` if you want to use `wsa:EndpointReferenceType` type elements in the request or response messages. End point references can be included in application data and you have the option of using the EPR in API commands such as **WSACONTEXT BUILD**. Setting the **WSADDR-EPR-ANY** parameter to `TRUE` indicates that CICS must not transform the EPR into a language structure at run time; instead, CICS treats the EPR data as an <xsd:any> element and stores it in a named container.

   This example WSDL fragment shows a <wsa:To> MAP being passed as an element of type `wsa:EndpointReferenceType`:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="exampleEPR" targetNamespace="http://example.ibm.com/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:s0="http://example.ibm.com/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  <types>
    <xs:schema targetNamespace="http://test.org/"
            xmlns:s="http://www.w3.org/2001/XMLSchema"
            xmlns:s0="http://example.ibm.com/"
            xmlns:wsa="http://www.w3.org/2005/08/addressing">
      ...
       <xs:element name="exampleResponse" type="s0:typeResponse"/>
       <xs:complexType name="typeResponse">
         <xs:sequence>
           <xs:element name="myEpr" type="wsa:EndpointReferenceType"/> 1
         </xs:sequence>
       </xs:complexType>
       ...
```

```
    </xs:schema>
  </types>
  ...
  <message name="msgResponse">
    <part element="s0:exampleResponse" name="response"/>
  </message>
  ...
</definitions>
```

When the element, `<xs:element name="myEpr" type="wsa:EndpointReferenceType"/>` **1**, is processed by DFHWS2LS with the **WSADDR-EPR-ANY** parameter set to `TRUE`, the `myEpr` element data is stored in a named container as an <xsd:any> element and a pointer to the container added to the generated language structure.

For example, the COBOL language structure generated by DFHWS2LS for the `myEpr` element is shown here:

```
09 myEpr.
   12 myEpr-xml-cont               PIC X(16).
   12 myEpr-xmlns-cont             PIC X(16).
```

The `myEpr-xml-cont` container stores the name of the container that contains the `myEpr` data. The `myEpr-xmlns-cont` is an optional container that is populated with any XML namespace declarations that are in scope.

4. Save and submit the DFHWS2LS job.

Results

CICS creates a web service binding to handle the data transformation and language structures that you can use to create the service requester or provider application.

What to do next

To enable the web service, perform a pipeline scan to create the required CICS resources.

Default end point references

Most WSDL documents contain the address at which the web service is hosted. In WS-Addressing, the WSDL document can also contain an end point reference (EPR) for the web service. This EPR can contain additional metadata to facilitate communication between the requester and provider applications.

If you use DFHWS2LS to process the WSDL, the EPR is saved in the web service binding and is used by CICS to send request and response messages. Any reference parameters, <wsa:ReferenceParameters>, that are defined in the EPR are included in the SOAP message. This EPR is known as the default EPR, because it can be overridden by the application. If the application does not supply an explicit EPR, the default EPR from the WSDL is used.

The following WSDL 1.1 fragment includes a default EPR: `<soap:address location="http://example.ibm.com:12345/exampleTest" />`. The <port> element includes a child element, <wsa:EndpointReference>, the address specified by the child element, **2**, must match the address specified by the parent element, **1**:

```
<service name="exampleService">
   <port name="examplePort" binding="s0:createBinding">
      <soap:address location="http://example.ibm.com:12345/exampleTest" /> 1

      <wsa:EndpointReference
```

```
                        xmlns:example="http://example.ibm.com/namespace"
                        xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
                        wsdli:wsdlLocation="http://example.ibm.com/location "
                        title="http://example.ibm.com/example/example_wsdl"
                        class="http://example.ibm.com/example/example.wsdl">
                        <wsa:Address>http://example.ibm.com:12345/exampleTest</wsa:Address> 2
                        <wsa:Metadata>
                           <wsam:InterfaceName>example:Inventory</wsam:InterfaceName>
                        </wsa:Metadata>
                        <wsa:ReferenceParameters>
                          <example:AccountCode>123456789</example:AccountCode>
                          <example:DiscountId>ABCDEFG</example:DiscountId>
                        </wsa:ReferenceParameters>
                    </wsa:EndpointReference>
                 </port>
              </service>
```

Explicit actions

WSDL documents can explicitly define the values of the <wsa:Action> properties. If the WSDL document does not contain explicitly defined <wsa:Action> properties, CICS builds default actions when the WSDL is processed by DFHWS2LS.

WSDL 1.1

The following WSDL 1.1 fragment represents a booking system that contains explicitly defined <wsa:Action> properties:

```
<definitions targetNamespace="http://example.ibm.com/namespace" ...>
   ...
   <portType name="bookingSystem">
      <operation name="makeBooking">
         <input message="tns:makeBooking"
            wsa:Action="http://example.ibm.com/namespace/makeBooking"
         </input>
         <output message="tns:bookingResponse"
             wsa:Action="http://example.ibm.com/namespace/bookingResponse"
         </output>
      </operation>
   </portType>
   ...
</definitions>
```

In this example, the input action of the `makeBooking` operation is explicitly defined as `http://example.ibm.com/namespace/makeBooking`, and the output action is explicitly defined as `http://example.ibm.com/namespace/bookingResponse`.

WSDL 2.0

The following WSDL 2.0 fragment represents a booking system that contains explicitly defined <wsa:Action> properties:

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <interface name="bookingInterface">
    <operation name="makeBooking" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input element="tns:makeBooking" messageLabel="In"
            wsa:Action="http://example.ibm.com/namespace/makeBooking"/>
      <output element="tns:makeBookingResponse" messageLabel="Out"
            wsa:Action="http://example.ibm.com/namespace/makeBookingResponse"/>
    </operation>
  </interface>
  ...
</description>
```

In this example, the input action of the `makeBooking` operation is explicitly defined as `http://example.ibm.com/namespace/makeBooking`, and the output action is defined as `http://example.ibm.com/namespace/makeBookingResponse`.

For more information, see the W3C WS-Addressing 1.0 Metadata specification.

Default actions for WSDL 1.1

If a WSDL 1.1 document does not contain explicitly specified <wsa:Action> properties, CICS builds default input, output, and fault actions when the WSDL is processed by DFHWS2LS.

Default input and output actions for WSDL 1.1

The following pattern is used by CICS in WSDL 1.1 documents that follow either the recommendation schema or the submission schema to construct a default input or output action:

```
[target namespace]/[port type name]/[input|output name]
```

Default fault actions for WSDL 1.1

If you are following the recommendation schema, the way that CICS builds the default fault action differs from the behavior described in the schema. The following pattern is used by CICS, in WSDL 1.1 documents that follow the recommendation schema, to construct a default fault message. Notice that the fault name is omitted.

```
[target namespace]/[port type name]/[operation name]/Fault/
```

If you are following the submission schema, the way that CICS builds the default fault action follows the behavior described in the schema. The following pattern is used by CICS, in WSDL 1.1 documents that follow the submission schema, to construct a default fault message:

```
[target namespace]/[port type name]/[operation name]/Fault/[fault name]
```

Example of the default actions generated by CICS for a WSDL 1.1 document

This example of a booking system illustrates how CICS constructs default actions from a WSDL 1.1 document:

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <portType name="bookingInterface">
    <operation name="makeBooking">
      <input element="tns:makeBooking" name="MakeBooking"/>
      <output element="tns:bookingResponse" name="BookingResponse"/>
         <fault message="tns:InvalidBooking" name="InvalidBooking"/>
    </operation>
  </interface>
  ...
</definitions>
```

The WSDL fragment has the following addressing properties:

Property name	Value
[targetNamespace]	`http://example.ibm.com/namespace`
[portType name]	`bookingInterface`
[operation name]	`makeBooking`

Property name	Value
[input name]	`MakeBooking`
[output name]	`BookingResponse`
[fault name]	`InvalidBooking`

The following actions are created from these values:

Action	Value
Input Action	`http://example.ibm.com/namespace/bookingInterface/MakeBooking` If the [input name] is not specified, the value of the [operation name] with "Request" appended is used instead. For example, in this case the Input Action is `http://example.ibm.com/namespace/bookingInterface/makeBookingRequest`.
Output Action	`http://example.ibm.com/namespace/bookingInterface/BookingResponse` If the [output name] is not specified, the value of the [operation name] with "Response" appended is used instead. For example, in this case the Output Action is `http://example.ibm.com/namespace/bookingInterface/makeBookingResponse`
Fault Action (Recommendation schema)	`http://example.ibm.com/namespace/bookingInterface/MakeBooking/Fault/` Notice that the [fault name] is omitted.
Fault Action (Submission schema)	`http://example.ibm.com/namespace/bookingInterface/MakeBooking/Fault/InvalidBooking`

For more information, see the W3C WS-Addressing 1.0 Metadata specification.

Default actions for WSDL 2.0

If a WSDL 2.0 document does not contain explicitly specified <wsa:Action> properties, CICS build default input, output, and fault actions when the WSDL is processed by DFHWS2LS.

Default input and output actions for WSDL 2.0

The following pattern is used by CICS, in WSDL 2.0 documents that follow the recommendation schema, to construct default actions for inputs and outputs:

```
[target namespace]/[interface name]/[operation name][direction token]
```

Default fault actions for WSDL 2.0

If you are following the recommendation schema, the way that CICS builds the default action for WS-Addressing faults differs from the behavior described in the schema. If you are following the submission schema, the way that CICS builds the default action for WS-Addressing faults follows the behavior described in the schema.

The following pattern is used by CICS, in WSDL 2.0 documents that follow the recommendation schema, to construct a default action for faults. Notice that the fault name is omitted.

```
[target namespace]/[interface name]/
```

The following pattern is used by CICS, in WSDL 2.0 documents that follow the submission schema, to construct a default action for faults:

```
[target namespace]/[interface name]/[fault name]
```

Example of the default actions generated by CICS for a WSDL 2.0 document

This example shows how CICS constructs default actions for a WSDL 2.0 document following the recommendation schema:

```
<description targetNamespace="http://example.ibm.com/namespace" ...>
  ...
  <interface name="bookingInterface">
    <operation name="makeBooking" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input element="tns:makeBooking" messageLabel="In"/>
      <output element="tns:bookingResponse" messageLabel="Out"/>
    </operation>
  </interface>
  ...
</definitions>
```

The WSDL fragment has the following addressing properties:

Property Name	Value
[targetNamespace]	`http://example.ibm.com/namespace`
[interface name]	`bookingInterface`
[operation name]	`makeBooking`
[direction token]	Either `Request` or `Response`.

The following input and output actions are created from these values:

Action	Value
Input Action	http://example.ibm.com/namespace/bookingInterface/makeBookingRequest
Output Action	http://example.ibm.com/namespace/bookingInterface/makeBookingResponse

For more information, see the W3C WS-Addressing 1.0 Metadata specification.

Message exchanges

Web Services Addressing (WS-Addressing) supports these message exchanges: one-way, two-way request-response, synchronous request-response, and asynchronous request-response.

Web Services Addressing message exchanges involve message addressing properties (MAPs) and endpoint references (EPRs).

At run time CICS ensures that the SOAP header of the request message contains the relevant WS-Addressing message information, the requester application does not have to set the WS-Addressing headers and might not even be aware that it is using WS-Addressing.

One-way

This straightforward one-way message is defined as an input-only operation. The web Services Description Language (WSDL) for this operation takes the following form:

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
</operation>
```

If you are using WS-Addressing, CICS adds the <wsa:Action> MAPs and the <wsa:MessageID> MAP to the SOAP message header of the WS-Addressing request message at run time to ensure compliance with the WS-Addressing specification.

The <wsa:MessageID> MAP is a unique ID, if not specified CICS generates this ID automatically.

The <wsa:Action> MAPs are derived from the WSDL and stored in the WSBind file.

You can override the values of these MAPs using the CICS WS-Addressing API commands.

Two-way request-response

This two-way exchange involves a request message and a response message. The response part of the operation can be defined as an output message, a fault message, or both. The WSDL definition for a request-response operation takes the following form:

```
<operation name="myOperation">
  <input message="tns:myInputMessage"/>
  <output message="tns:myOutputMessage"/>
  <fault="tns:myFaultMessage"/>
</operation>
```

Responses to, or faults generated from, requests that are directed at endpoints are targeted at the <wsa:ReplyTo> MAP or the <wsa:FaultTo> MAP depending on whether the reply type is normal or a fault.

Specify a <wsa:ReplyTo> or <wsa:FaultTo> MAP in the request message to indicate where the response must be sent.

If you are using the recommendation specifications and do not specify a value for the <wsa:ReplyTo> MAP, the <wsa:ReplyTo> MAP defaults to an endpoint reference that contains the anonymous URI (http://www.w3.org/2005/08/addressing/anonymous), which causes CICS to send the response back to the requester.

If you are using the recommendation specifications and do not specify a value for the <wsa:FaultTo> MAP, the <wsa:FaultTo> MAP defaults the value of the <wsa:ReplyTo> MAP.

If the requester builds MAPs that are incorrect and that cause validation failures, CICS sends the fault message back to the requester instead of to the address specified by the <wsa:FaultTo> MAP.

Synchronous request-response

By default, the response part of a two-way message is returned according to the underlying protocol in use. In the case of an HTTP request, the response is returned synchronously in the HTTP response.

Asynchronous request-response

An asynchronous response is targeted at another web service and does not arrive back at the original requester application. In the case of an HTTP request, the connection with the requesting client is closed with an HTTP 202 response. If the web service provider is running on a CICS system, the requester application will receive an empty response message. If the web service provider is running on a WebSphere MQ system, the requester application will not receive any response.

To change the destination of the response part of a two-way message, you must specify the appropriate addresses in the <wsa:ReplyTo> MAP, or the <wsa:ReplyTo> and <wsa:FaultTo>, MAPs.

For a full list of the MAPs that are mandatory in WSDL 1.1 and WSDL 2.0, see "Mandatory message addressing properties for WS-Addressing" on page 361.

Related concepts:

"WSDL and message exchange patterns" on page 30
A WSDL 2.0 document contains a message exchange pattern (MEP) that defines the way that SOAP 1.2 messages are exchanged between the web service requester and web service provider.

Related reference:

"Mandatory message addressing properties for WS-Addressing" on page 361
The WS-Addressing 1.0 metadata specification states which message addressing properties (MAPs) must be included in WSDL 1.1 and WSDL 2.0 documents. The CICS implementation of WS-Addressing helps you to comply with the WS-Addressing specifications by automatically supplying values for these mandatory MAPs.

Mandatory message addressing properties for WS-Addressing

The WS-Addressing 1.0 metadata specification states which message addressing properties (MAPs) must be included in WSDL 1.1 and WSDL 2.0 documents. The CICS implementation of WS-Addressing helps you to comply with the WS-Addressing specifications by automatically supplying values for these mandatory MAPs.

You can specify your own values for MAPs in the WSDL that you supply, and you can update these values in the addressing context using the CICS WS-Addressing API commands. If you do not supply values for the mandatory MAPs, CICS will generate values for you.

The following table lists which MAPs are mandatory for the different supported message exchange patterns (MEPs) with WSDL 1.1 and WSDL 2.0:

Table 20. Mandatory message addressing properties for WS-Addressing.

WS-Addressing MAP name	Description	Mandatory in WSDL 1.1	Mandatory in WSDL 2.0
<wsa:To>	The address of the intended receiver of the message.	No	No
<wsa:Action>	The WS-Addressing action: input, output, or fault.	Mandatory for the following MEPs: One-way Two-way (Request) Two-way (Response)	Mandatory for the following MEPs: In-only Robust In-only (In) Robust In-only (Fault) In-out (In) In-out (Out) In-optional-out (In) In-optional-out (Out)
<wsa:From>	The endpoint from which the message originated.	No	No
This	value	not	required
<wsa:ReplyTo>	The endpoint of the intended receiver for replies to the message.	No	No
<wsa:FaultTo>	The endpoint of the intended receiver for faults related to the message.	No	No
<wsa:MessageID>	A unique message identifier.	Mandatory for the following MEPs: Two-way (Request)	Mandatory for the following MEPs: Robust In-only (In) In-out (In) In-optional-out (In)
<wsa:RelatesTo>	A pair of values that indicate how this message relates to another message. This element includes the <wsa:MessageID> of the related message and an optional attribute conveys the relationship type.	Mandatory for the following MEPs: Two-way (Response)	Mandatory for the following MEPs: Robust In-only (Fault) In-out (Out) In-optional-out (Out)

For more information, see the *W3C WS-Addressing 1.0 Metadata* specification: http://www.w3.org/TR/ws-addr-metadata.

Notes:

- If a value is not set for the address element of the <wsa:ReplyTo> MAP, the address is set to the anonymous URI: `http://www.w3.org/2005/08/addressing/anonymous`. The anonymous URI indicates that responses are sent back to the requester.
- If a value is not specified for the address element of the <wsa:FaultTo> MAP, CICS sets this address to the same value as the address element of the <wsa:ReplyTo> MAP.

Note that if the requester builds MAPs that are incorrect and which cause validation failures, CICS sends the fault message back to the requester instead of to the address specified by the <wsa:FaultTo> MAP.

- If the value of the <wsa:To> MAP is not specified, CICS set the address to the anonymous URI: `http://www.w3.org/2005/08/addressing/anonymous`. The anonymous URI indicates that the request is to be sent to the address specified in the DFHWS-URI container; for more information, see "DFHWS-URI container" on page 304.
- You can define the <wsa:Action> MAPs explicitly in your WSDL document, or you can let CICS generate them automatically.
- CICS automatically sets a unique value for the <wsa:MessageID> MAP at run time for request messages that expect a response, and for response messages.
- The <wsa:RelatesTo> MAP is mandatory for response messages. The relationship type of the message is optional and defaults to http://www.w3.org/2005/08/addressing/reply.

Related concepts:

"Message exchanges" on page 359
Web Services Addressing (WS-Addressing) supports these message exchanges: one-way, two-way request-response, synchronous request-response, and asynchronous request-response.

Related reference:

"DFHWS-URI container" on page 304
DFHWS-URI is a container of DATATYPE(CHAR) that contains the URI of the service.

Web Services Addressing security

Communications traveling on a public network using Web Services Addressing (WS-Addressing) must be adequately secured and a sufficient level of trust must be established between the communicating parties. You are recommended to use transport level security, such as SSL or HTTPS, to secure your communications.

Transport level security, such as SSL or HTTPS, is the most straightforward way to ensure that your WS-Addressing communications are secure. If transport level security is not available, you can secure your messages by signing the WS-Addressing message addressing properties and encrypting the endpoint references.

CICS cannot sign headers containing WS-Addressing message addressing properties or encrypt endpoint references. However, CICS can verify signatures on incoming messages and can decrypt headers that have been encrypted. If you want to use signing and encryption to secure your communications, you must use an external security gateway, such as the IBM WebSphere DataPower XML Security Gateway. For more information, see IBM WebSphere DataPower XML Security Gateway.

Web Services Addressing example

This example provides a high-level overview of the process that takes place when a customer places an order with a company that uses Web Services Addressing to send messages.

An international company that sells electronic components uses Web Services Addressing in its business. The infrastructure of this company consists of an Ordering Client, a group of Distribution Services, a Fulfilment Service, and a Configuration Service.

Using WS-Addressing offers the company the following benefits:

- WS-Addressing provides a transport-independent mechanism for transferring messages, this encourages interoperability between web services running on different platforms. In this example, the distribution services owned by the company are running on a variety of platforms; WS-Addressing makes interoperability between different platforms straightforward because the web service requesters and providers do not need to be aware of the platform on which the service that they are exchanging messages with is running.
- WS-Addressing can be used to change the destination of the reply message by updating the EPR in the <wsa:ReplyTo> MAP. In this example, the Fulfilment Service modifies the destination of the response message when it selects the Distribution Service to which the message is diverted.

The company has several distribution centers in a number of different countries; each of the distribution centers is represented in this example by a Distribution Service and is registered with the Configuration Service.

The Fulfilment Service selects which Distribution service is the most appropriate to process the order based on a variety of factors, which might include the availability of items requested and the distance of the Distribution Center from the customer.

Addressing information is passed to and from the Configuration Service. The Configuration Service stores the addresses of the available services in the form of Endpoint References. New services register with the Configuration Service by creating an EPR using the **WSAEPR CREATE** command and sending the EPR to the Configuration Service. The Configuration Service requires the EPR as a block of XML, so the **WSADDR-EPR-ANY** parameter on DFHWS2LS must be set to `TRUE`. The **WSADDR-EPR-ANY=TRUE** option is used to instruct CICS to treat the EPR as an <xsd:any> element; CICS must place it in a container instead of transforming it into a language structure at run time.

The way in which these services interact is shown in the following diagram. The diagram shows other services, which have been excluded from the task, that might be relevant in a business application:

- A Tracking Service, which can be updated by each of the other services with the status of the order.
- A Problem Resolution service to handle any fault messages that arise.
- An Ordering Client callback service to handle any reply messages directed at the Ordering Client.

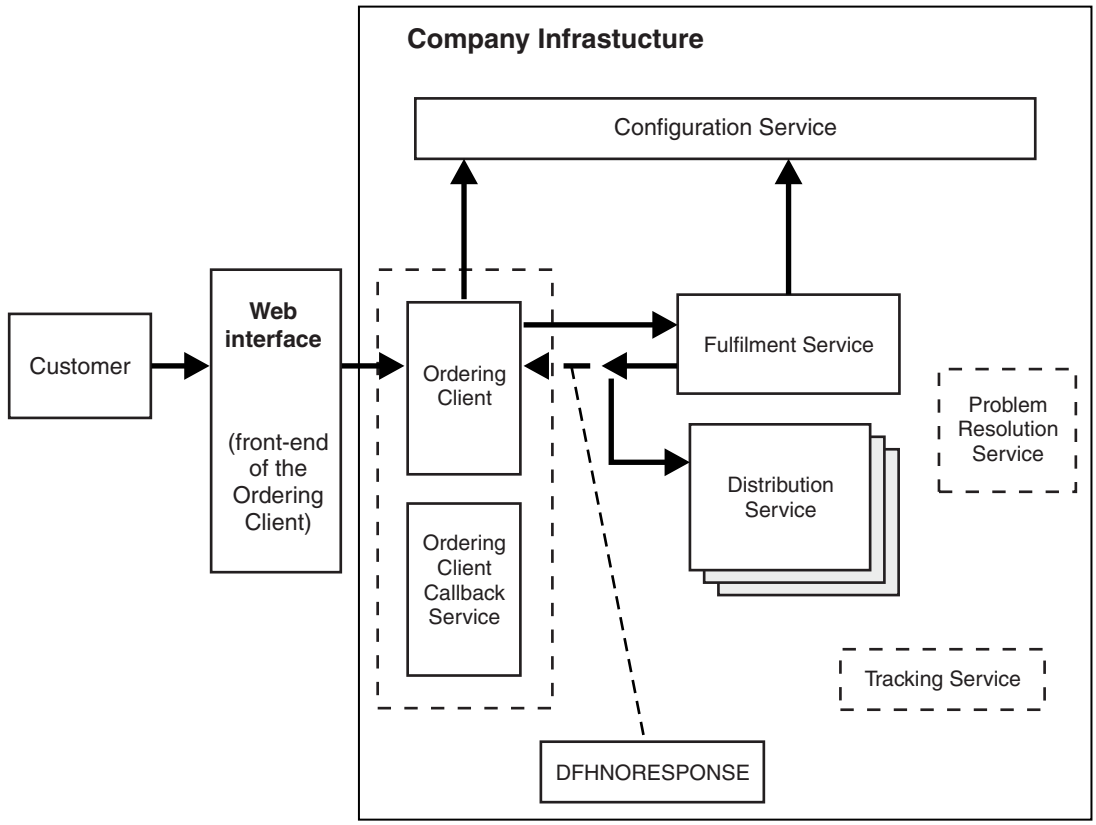


Figure 36. Company infrastructure

The following steps describe the process that takes place from the time a customer places an order to the point at which that order is processed.

1. A customer places an order with the company.
   a. The customer places the order on the company website, which is the front end for the Ordering Client.
   b. The Ordering Client takes the customer's contact details as part of the order.
   c. The Ordering Client returns a confirmation and a unique order reference to the customer through the web interface.
2. The Ordering Client sends the order request to the Fulfilment Service.
   a. If the Ordering Client does not already know the EPR for the Fulfilment Service, it requests it from the Configuration Service. The process involved when the Ordering Client requests the EPR of the Fulfilment Service from the Configuration service is detailed in the Example of <wsa:To> section.
   b. The Ordering Client issues the **INVOKE SERVICE** command for the Fulfilment Service. WS-Addressing routes the message to the address specified by the To EPR in the request addressing context.
3. The Fulfilment Service selects a Distribution Service to process the order and redirects the response message to that service.
   a. The Fulfilment Service uses a **WSACONTEXT GET** command to extract the order reference and other addressing properties from the addressing context.
   b. The Fulfilment Service selects the most appropriate Distribution Service from the Configuration Service.
   c. The <wsa:ReplyTo> EPR is added to the addressing context:

```
<wsa:EndpointReference
     xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsa:Address>http://www.example.ibm.com/DistributionService</wsa:Address>
  </wsa:EndpointReference>
```

The Fulfilment Service uses the **WSACONTEXT BUILD** command to add the ReplyTo EPR of the chosen Distribution Service to the request addressing context.

d. The Fulfilment Service uses the **WSACONTEXT BUILD** command repeatedly to add the order reference and other information to the request addressing context.

e. A DFHNORESPONSE container is added to the Ordering Client pipeline to indicate to the Ordering Client that it will not receive a response and the response message is redirected in the form of a request message to the Distribution Service.

4. The Distribution Service receives the redirected response message and processes the order.
   a. The Distribution Service uses a **WSACONTEXT GET** command to extract the order reference and addressing details from the request addressing context.
   b. The Distribution Service process the order.

Example of <wsa:To>

1. The Ordering Client requests the EPR of the service that it wants to send a message to from the Configuration Service. In this example, the Ordering Client requests the EPR of the Fulfilment Service.
2. The Configuration Service creates and sends a response message:
   a. The Configuration Service creates the requested <wsa:To> EPR for the Fulfilment Service using the **WSAEPR CREATE** API command: `EXEC CICS WSAEPR CREATE`.
   b. The Configuration Service writes the output from the **WSAEPR CREATE** command to a container: `EXEC CICS PUT CONTAINER(work-cont)`.
   c. The Configuration Service copies the container name into the `myEpr-xml-cont` element: `MOVE work-cont TO myEpr-xml-cont`.
   d. The Configuration Service sends a response message to the Ordering Client, this message contains the contents of the container named by the `myEpr-xml-cont` container. In this example, the contents of the `work-cont` container is sent to the Ordering Client inside the `<wsa:myEpr>` element:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   ...
   <env:Body>
      <wsa:myEpr>
         <wsa:EndpointReference>
            <wsa:Address>
          Fulfilment_Service_EPR_XML
            </wsa:Address>
         </wsa:EndpointReference>
      </wsa:myEpr>
   </env:Body>
   ...
</env:Envelope>
```

Figure 32 on page 367 shows the request-response message exchange between the Ordering Client and the Configuration Service. This message exchange involves two typical web services pipelines.

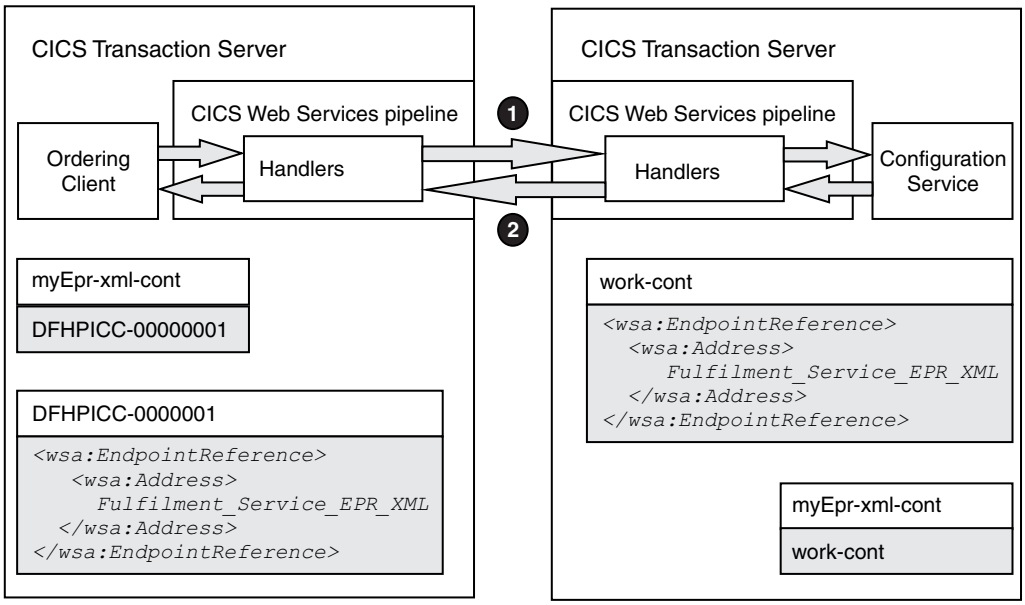


Figure 37. Request-response message exchange between the Ordering Client and the Configuration Service

3. The Ordering Client receives the response message, builds the <wsa:To> EPR, and sends a request to the Fulfilment Service:
   a. The Ordering Client extracts the <wsa:To> EPR data from the response message.
   b. CICS populates a unique container, in this example the `DFHPICC-00000001` container, with the <wsa:To> EPR data.
   c. CICS copies the name of the container, in this example `DFHPICC-00000001`, into the `myEpr-xml-cont` element.
   d. The Ordering Client reads the contents of the container specified by the `myEpr-xml-cont` element and provides it as input to the **WSACONTEXT BUILD** API command. The **WSACONTEXT BUILD** command uses this input to build the <wsa:To> EPR for the Fulfilment Service.
   e. The Ordering Client issues an **INVOKE SERVICE** command which initiates the pipeline processing.
   f. The CICS web services addressing handler, DFHWSADH, on the outbound pipeline converts the <wsa:To> EPR into an address and an optional set of reference parameters which it puts into the header of the SOAP request message that is being sent to the Fulfilment Service:

```
<env:Header>
  <wsa:To>http://example.ibm.com/Fulfilment_Service</wsa:To>
</env:Header>
```

Figure 33 on page 368 shows the request from the Ordering Client to the Fulfilment service. This request involves a web services pipeline that includes the CICS web services addressing handler, DFHWSADH.

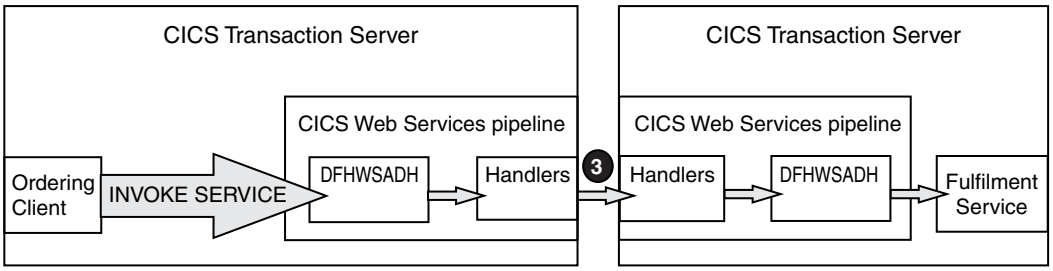


Figure 38. Request from the Ordering Client to the Fulfilment Service

Related reference:

"Web Services Addressing terminology" on page 368
Terms used to explain Web Services Addressing (WS-Addressing) support.

Web Services Addressing terminology

Terms used to explain Web Services Addressing (WS-Addressing) support.

addressing context
An XML document that stores WS-Addressing message addressing properties (MAPs) before they are sent in SOAP request messages and after they are received from SOAP request and response messages.

endpoint reference (EPR)
An XML structure containing addressing information that is used to route a message to a web service. This addressing information includes the destination address of the message, optional reference parameters for use by the application, and optional metadata.

message addressing property (MAP)
An XML element that conveys addressing information for a specific web service message, such as a unique message ID, the destination of the message, and the endpoint references of the message.

Chapter 13. Support for securing web services

CICS Transaction Server for z/OS provides support for a number of related technologies that you can use to secure SOAP and JSON messages.

Some of these technologies are available as part of the HTTP protocol, and are equally applicable to both SOAP and JSON. Some use the *Web Services Security (WSS): SOAP Message Security 1.0* specification, and are only available for SOAP. For information on the shared TCP/IP and HTTP security options, see Security for TCP/IP clients and Security for CICS web support in Securing.

For information about using SAML assertions, see Overview of SAML support.

SOAP web services security

Web Services Security (WSS): SOAP Message Security 1.0 describes the use of *security tokens* and *digital signatures* to protect and authenticate SOAP messages. For more information, see the WSS: Soap Message Security 1.0 specification.

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 in the body or the header. You can give different levels of protection to different elements in the SOAP message.

The *Web Services Trust Language* specification enhances Web Services Security further by providing a framework for requesting and issuing security tokens, and managing trust relationships between web service requesters and providers. This extension to the authentication of SOAP messages enables web services to validate and exchange security tokens of different types byusing a trusted third party. This third party is called a *Security Token Service* (STS). For more information about the Web Services Trust Language, see the WS-Trust Language specification.

CICS Transaction Server for z/OS provides support for these specifications by using a CICS-supplied security handler in the pipeline:

- For outbound messages, CICS provides support for digital signing and encryption of the entire SOAP body. CICS can also exchange a username token for a security token of a different type with an STS.
- For inbound messages, CICS supports messages in which the body, or elements of the body and header, are encrypted or digitally signed. CICS can also exchange and validate security tokens with an STS.

CICS also provides a separate Trust client interface so that you can interact with an STS without using the CICS security handler.

Note: Web Services Security is potentially not conformant with SP800-131A. Web Services Security is configured by adding a handler into the pipeline and CICS has no control over the processing in a customer-written handler. If you use digital signatures, you can specify only the algorithms `dsa-sha1` and `rsa-sha1`. These algorithms are not SP800-131A-conformant. The two-key tripledes encryption algorithm, which can be used to encrypt a SOAP body, is also non-conformant.

Prerequisites for Web Services Security

To implement Web Services Security, you must apply these updates to your CICS region: install the IBM XML Toolkit for z/OS v1.10, apply APAR OA14956, and add 3 libraries to the DFHRPL concatenation.

About this task

Complete the following steps before you implement Web Services Security:

Procedure

1. Install the free IBM XML Toolkit for z/OS v1.10. You can download it from the following site: http://www.ibm.com/servers/eserver/zseries/software/xml/. You must install version 1.10. Later versions do not work with Web Services Security support in CICS.
2. Apply ICSF APAR OA14956 if it is not already installed in z/OS.
3. Add the following libraries to the DFHRPL concatenation:
   - *hlq*.SIXMLOD1, where *hlq* is the high-level qualifier of the XML Toolkit.
   - *hlq*.SCEERUN, where *hlq* is the high-level qualifier of the Language Environment.
   - *hlq*.SDFHWSLD, where *hlq* is the high-level qualifier of the CICS installation; for example CICSTS53.

   The first two libraries contain DLLs that are required at run time by the security handler. IXM4C57 is provided by the XML Toolkit and is found in *hlq*.SIXMLOD1; C128N is provided by the Language Environment run time and is found in *hlq*.SCEERUN.

   The *hlq*.SDFHWSLD library enables CICS to find the DFHWSSE1 and DFHWSXXX Web Services Security modules.
4. You might need to increase the value of the **EDSALIM** system initialization parameter. The three DLLs that are loaded require approximately 15 MB of EDSA storage.

Results

If you do not have the libraries specified, you see the following message:

```
CEE3501S The module module_name was not found.
```

The *module_name* varies depending on which library is missing.

Planning to secure SOAP web services

You can decide the best way to secure your web services. CICS supports a number of options, including a configurable security message handler and a separate Trust client interface.

About this task

CICS implements Web Services Security (WS-Security or WSS) at a pipeline level, rather than for each web service. Answer the following questions to decide how best to implement security.

Procedure

1. Is the performance of your pipeline processing important? The use of WSS to secure your web services incurs a significant performance impact.

   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 only those sections that need to be confidential, you derive the following benefits:

   - You do not incur the overhead of encrypting and decrypting at every node in a chain of intermediate processes.
   - You can route a confidential message over a public network of untrusted nodes, where only the ultimate recipient of the data can understand it.

   As an alternative to using WSS, you can use SSL to encrypt the whole data stream.
2. If you want to use WSS, what level of security do you want? The options range from basic authentication, where the message header includes a user name and a password, through to combining digital signatures and encryption in the message. The options that the CICS security handler supports are described in "Options for securing SOAP messages."
3. Does the CICS-supplied security handler meet your requirements? If you want to perform more advanced security processing, you must write your own custom security handler. This handler must perform the necessary authentication of messages, either directly with RACF or using a Security Token Service, and handle the processing of digital certificates and encrypted elements. See "Writing a custom security handler" on page 586 for details.
4. Does your pipeline include an MTOM handler? If you are planning to enable both the MTOM handler and the security handler in your pipeline configuration file, any MIME Multipart or Related messages are processed in compatibility mode, because the security handler cannot parse the XOP elements in the body of the message. This processing can have a further effect on the performance of the pipeline processing.

Options for securing SOAP messages

CICS supports both the signing and encryption of 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.

Signing and encryption of SOAP messages are not supported for provider mode Axis2 web service Java applications or for provider web services that attach to the pipeline using Axis2 MessageContext.

You can choose from these options:

Trusted authentication

In service provider pipelines, CICS can accept a username token in the SOAP message header as trusted. This type of security token typically contains a user name and password, but in this case the password is not required. CICS trusts the provided user name and places it in container DFHWS-USERID, and the message is processed in the pipeline.

In service requester pipelines, CICS can send a username token without the password in the SOAP message header to the service provider.

Basic authentication
In service provider mode, CICS can accept a username token in the SOAP message header for authentication on inbound SOAP messages. This type of security token contains a user name and password. CICS verifies the username 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 cannot verify the username token, a SOAP fault message is returned to the service requester.

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

HTTP basic authentication
In service provider mode, CICS can accept basic authentication information over an HTTP protocol. The service requester uses a URIMAP definition to specify that credentials (user identification information) can be captured by the global user exit, XWBAUTH. XWBAUTH passes this information to CICS on request and CICS sends the information in an HTTP authorization header to the service provider.

Advanced authentication
In service provider and requester pipelines, you can verify or exchange security tokens with a Security Token Service (STS) for authentication purposes. This authentication enables CICS to accept and send messages that have security tokens in the message header that are not normally supported; for example, Kerberos tokens or SAML assertions.

For an inbound message, you can select to verify or exchange a security token. If the request is to exchange the security token, CICS must receive a username token back from the STS. For an outbound message, you can exchange a username token only for a security token.

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 type of security token 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 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. With this second certificate, you can 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 to assert that work runs 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 a *symmetric key*. It is then included in the message and encrypted using a combination of the public key of the intended recipient and the asymmetric key encryption algorithm RSA 1.5. This encryption provides you with increased security, because the asymmetric algorithm is complex and it is difficult to decrypt the symmetric key. However, you obtain better performance because the majority of the SOAP message is encrypted with the symmetric algorithm, which is faster to decrypt.

For inbound SOAP messages, you can encrypt an element in the SOAP body and then encrypt the SOAP body as a whole. This sort of encryption 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 sort of encryption is not supported for outbound SOAP messages.

CICS does not support inbound SOAP messages that have an encrypted element in the message header only 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.

ICRX-based identity propagation
In service provider mode, you can use an unauthenticated ICRX (Extended Identity Context Reference) identity token in the same circumstances that you would use an unauthenticated WS-Security user ID token. An ICRX identity token is a z/OS identifier that maps to a user ID. CICS resolves the ICRX identity token to a user ID and places a copy in the DFHWS-ICRX container. CICS also populates the DFHWS-USERID container. For more information about an ICRX identity token, see Identity propogation and distributed security in Securing.

Authentication using a Security Token Service

CICS can interoperate with a Security Token Service (STS), such as Tivoli Federated Identity Manager, to provide more advanced authentication of web services.

An STS is a web service that acts as a trusted third party to broker trust relationships between a web service requester and a web service provider. In a similar manner to a certificate authority in an SSL handshake, the STS guarantees that the requester and provider can "trust" the credentials that are provided in the message. This trust is represented through the exchange of security tokens. An STS can issue, exchange, and validate these security tokens, and establish trust relationships, allowing web services from different trust domains to communicate successfully. For more details, see the Web Services Trust Language specification.

CICS acts as a Trust client and can send two types of web service request to an STS. The first type of request is to validate the security token in the WS-Security message header; the second type of request is to exchange the security token for a different type. These requests enable CICS to send and receive messages that contain different security tokens from a wide variety of trust domains, such as SAML assertions and Kerberos tokens.

You can either configure the CICS security handler to define how CICS interacts with an STS or write your own message handler to use a separately provided Trust client interface. Whichever method you select, use SSL to secure the connection between CICS and the STS.

How the security handler calls the STS

The CICS security handler uses the information in the pipeline configuration file to send a web service request to the Security Token Service (STS). The type of request that is sent depends on the action that you want the STS to perform.

In a service provider pipeline

In a service provider pipeline, the security handler supports two types of actions, depending on the way you configure the security handler:

- Send a request to the STS to validate the first instance of a security token, or the first security token of a specific type, in the WS-Security header of the inbound message.
- Send a request to the STS to exchange the first instance of a security token, or the first security token of a specific type, in the WS-Security header of the inbound message, for a security token that CICS can understand.

The security handler dynamically creates a pipeline to send the web service request to the STS. This pipeline exists until a response is received from the STS, after which it is deleted. If the request is successful, the STS returns an identity token or the status of the validity of the token. The security handler places the RACF ID that is derived from the token in the DFHWS-USERID container.

If the STS encounters an error, it returns a SOAP fault to the security handler. The security handler then passes a fault back to the web service requester.

In a service requester pipeline

In a service requester pipeline, the security handler can request only to exchange a token with the STS. The pipeline configuration file defines what type of token the STS issues to the security handler.

If the request is successful, the RACF ID is placed in the DFHWS-USERID container and the token is included in the outbound message header. If the STS encounters an error, it returns a SOAP fault to the security handler. The security handler then passes the fault back through the pipeline to the web service requester application.

The security handler can request only one type of action from the STS for the pipeline. It can also exchange only one type of token for an outbound request message, and is limited to handling the first token in the WS-Security message header, either the first instance or the first instance of a specific type. These options cover the most common scenarios for using an STS, but might not offer you the processing that you require for handling inbound and outbound messages.

If you want to provide more specific processing to handle many tokens in the inbound message headers or exchange multiple types of tokens for outbound messages, use the Trust client interface. Using this interface, you can create a custom message handler to send your own web service request to the STS.

The Trust client interface

The Trust client interface enables you to interact with a Security Token Service (STS) directly, rather than using the security handler. In this way, you have the flexibility to provide more advanced processing of tokens than the processing offered by the security handler.

The Trust client interface is an enhancement to the CICS-supplied program DFHPIRT. This program is usually used to start a pipeline when a web service requester application has not been deployed using the CICS web services assistant. But it can also act as the Trust client interface to the STS.

You can invoke the Trust client interface by linking to DFHPIRT from a message handler or header processing program, passing a channel called DFHWSTC-V1 and a set of security containers. Using these containers, you have the flexibility to request either a validate or issue action from the STS, select which token type to exchange, and pass the appropriate token from the message header. DFHPIRT dynamically creates a pipeline, composes a web service request from the security containers, and sends it to the STS.

DFHPIRT waits for the response from the STS and passes this back in the DFHWS-RESTOKEN container to the message handler. If the STS encounters an error, it returns a SOAP fault. DFHPIRT puts the fault in the DFHWS-STSFAULT container and returns to the linking program in the pipeline.

You can use the Trust client interface without enabling the security handler in your service provider and service requester pipelines, or you can use the Trust client interface in addition to the security handler.

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.

The *WSS: SOAP Message Security* specification 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 is appropriate to sign elements that confirm receipt of an order, because these elements might have legal status. However, to avoid the overhead of signing the entire message, other information might safely remain unsigned.

For inbound messages, the security message handler can verify the digital signature on individual elements in the SOAP `<Header>` and the `<Body>`:

- Signed elements it encounters in the `<Header>`.
- Signed elements in the SOAP `<Body>`. If the handler is configured to expect a signed body, CICS rejects any SOAP message in which the body is not signed and issues a SOAP fault.

For outbound messages, the security message handler 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 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) Supported on inbound SOAP messages only.	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

Example of a signed SOAP message

This is example shows 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/xmldsig#"
                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/xmlenc#" 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">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHQjEMMAoGA1UEChMD
                                  SUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
                                  MDELMAkGA1UEBhMCR0IxDDAKBgNVBAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkq
                                  hkiG9w0BAQEFAAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
                                  6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8JXPFzA+BqV63JqRgJyxN6
                                  msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhC
                                  AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgG
                                  ZQVRFU0BVSy5JQk0uQ09ggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO
 </wsse:BinarySecurityToken>
 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
   <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
    <c14n:InclusiveNamespaces xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="ds wsu xenc SOAP-ENV "/>
   </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+gpafluShspHxhrjaFlXE=</ds:DigestValue> 3
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>drDH0XESiyN6YJm27mfK1ZMG4Q4IsZqQ9N9V6kEnw2lk7aM3if77XNFnyKS4deglbC3ga11kkaFJ 4
                     p4jLOmYRqqycDPpqPm+UEu7mzfHRQGe7H0EnFqZpikNqZK5FF6fvYlv2JgTDPwrOSYXmhzwegUDT
                     lTVjOvuUgXYrFyaO3pw=</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 encoding includes the public key that the intended recipient of the SOAP message uses 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 is 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.

The *WSS: SOAP Message Security* specification allows some of the contents of the `<Header>` element and all of the contents of the `<Body>` element 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 is appropriate to encrypt an individual's credit card details 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 remain unencrypted.

For inbound messages, the CICS-supplied security message handler can decrypt individual elements in the SOAP `<Body>`, and can decrypt elements in the SOAP `<Header>` if the SOAP body is also encrypted. The security message handler always decrypts these elements:

- Elements it encounters in the `<Header>` element in the order in which the elements are found.
- Elements in the SOAP `<Body>` element. 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, the security message handler supports encryption of the contents of the SOAP `<Body>` only; it does not encrypt any elements in the `<Header>` element. When the security message handler encrypts the `<Body>` element, all elements in the body are encrypted with the same algorithm and using the same key. The algorithm, and information about the key, are specified in the configuration information about the handler.

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 example of a SOAP message 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>
 <wsse:Security xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                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/xmlenc#" SOAP-ENV:mustUnderstand="1">

  <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" 1
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
                wsu:Id="x509cert00">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHQjEMMAoGA1UEChMD
                                   SUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
                                   MDELMAkGA1UEBhMCR0IxDDAKBgNVBAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkq
                                   hkiG9w0BAQEFAAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
                                   6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8JXPFzA+BqV63JqRgJyxN6
                                   msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhC
                                   AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgG
                                   A1UdEQQxMC+BEVdZQVRFU0BVSy5JQk0uQ09NggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO
                                   BgNVHQ8BAf8EBAMCAfYwHQYDVR0OBBYEFMiPX6VZKP5+mSOY1TLNQGVvJzu+MA0GCSqGSIb3DQEB
                                   BQUAA4GBAHdrS409Jhoe67pHL2gs7x4SpV/NOuJnn/w25sjjop3RLgJ2bKtK6RiEevhCDim6tnYW
                                   NyjBL1VdN7u5M6kTfd+HutR/HnIrQ3qPkXZK4ipgC0RWDJ+8APLySCxtFL+J0LN9Eo6yjiHL68mq
                                   uZbTH2LvzFMy4PqEbmVKbmA87alF
  </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>M6bDQtJrvX0pEjAEIcf6bq6MP3ySmB4TQOa/B5UlQj1vWjD56V+GRJbF7ZCES5ojwCJHRVKW1ZB5 4
                      Mb+aUzSWlsoHzHQixc1JchgwCiyIn+E2TbG3R9m0zHD3XQsKTyVaOTlR7VPoMBd1ZLNDIomxjZn2
                      p7JfxywXkObcSLhdZnc=</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>kgvqKnMcgIUn7rl1vkFXF0g4SodEd3dxAJo/mVN6ef211B1MZelg7OyjEHf4ZXwlCdtOFebIdlnK 6
                     rrksql1Mpw6So7ID8zav+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 encoding 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 begin

Before you perform this task, you must have RACF set up to work with CICS. Specify the **DFLTUSER**, **KEYRING**, and **SEC=YES** system initialization parameters in the CICS region that contains your web services pipelines.

Note: Multiple certificates with the same Distinguished Name on the same **KEYRING** are not supported.

Procedure

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. 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 (the certificate label in RACF), a combination of the ISSUER and SERIAL number, or the SubjectKeyIdentifier. CICS can recognize the SubjectKeyIdentifier only if it 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 must 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 recognize the SubjectKeyIdentifier only if it 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 must 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. The public key is defined in the pipeline configuration file.

What to do next

This configuration for signing and encrypting outbound messages requires that the certificate used is owned by the CICS region user ID. The certificate must be owned by the CICS region userid because RACF allows only the certificate owner to extract the private key, which is used for the signing or encryption process.

If CICS needs to sign or encrypt a message using a certificate that it does not own, you can share a single certificate between CICS systems by following the instructions in Using an existing certificate that is not owned by the CICS region user ID.

Configuring provider mode web services for identity propagation

Identity propagation with a web service request relies on trust-based configurations; for example, using a client-certified SSL connection from WebSphere DataPower. In this task, you configure a PIPELINE resource to expect an ICRX identity token in the WS-Security header, sent from a trusted client.

Before you begin

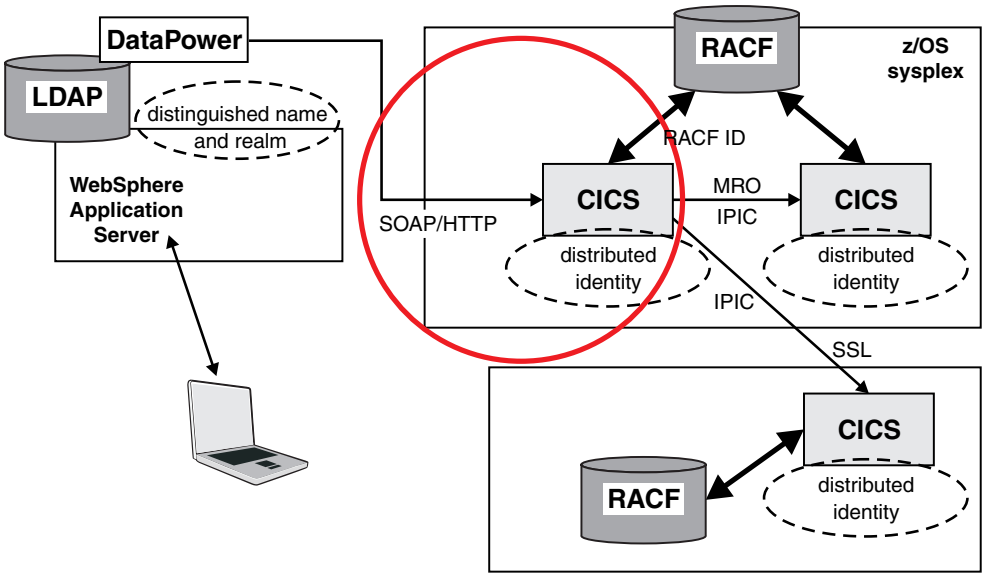
You must configure your RACF RACMAP settings before you configure your web service connections, otherwise you receive the RACF ICH408I message for every unmapped request that is sent to RACF. For more information about configuring the RACF **RACMAP** command, see Configuring RACF for identity propagation.

You must configure a *trust* relationship between the WebSphere DataPower appliance and CICS, for example, using SSL client certification between WebSphere DataPower and CICS. The digital certificate that WebSphere DataPower uses to identify itself must be associated with a user ID, and that user ID must be granted surrogate authority to assert identities. For more information about surrogate authority, see Surrogate user security.

About this task

This task explains how to use CICS with a WebSphere DataPower appliance to provide a web service configuration that can propagate distributed identities in a secure and robust way. The circle in the diagram indicates that this task explains the CICS-specific configuration.

Figure 39. Configuring CICS to expect an ICRX identity token from WebSphere DataPower.



WebSphere DataPower acts as an intermediary between CICS and other applications. Remote web service requester applications connect to the WebSphere DataPower appliance using the SOAP protocol. WebSphere DataPower authenticates the credentials supplied by the remote client and mapping the credentials to a z/OS ICRX identity token, which identifies the distributed identity of a user. The SOAP message is then forwarded to CICS over the trusted SSL connection with an ICRX identity token in a WS-Security header. For more information about ICRX identity tokens, see z/OS Security Server RACF Data Areas.

CICS receives the SOAP message from WebSphere DataPower. The PIPELINE configuration file specifies *blind* trust, because the only possible client is the WebSphere DataPower appliance, and WebSphere DataPower is communicating with CICS over a secure SSL connection. Therefore, you do not need to specify additional authentication in the PIPELINE configuration file. The WS-Security handler program locates the first ICRX found in the WS-Security header and uses the ICRX to identify the user.

Procedure

1. Create a PIPELINE resource, or edit an existing PIPELINE resource to specify the basic-ICRX mode, which allows the PIPELINE to receive an ICRX. The most typical combination is the blind trust with the basic-ICRX mode. For more information about the PIPELINE resource element, see "The <authentication> element" on page 265.

   Here is an example PIPELINE configuration file, showing blind trust with the basic-ICRX mode:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
   <service>
     <service_handler_list>
       <wsse_handler>
         <dfhwsse_configuration version="1">
           <authentication trust="blind" mode="basic-ICRX"/>
         </dfhwsse_configuration>
       </wsse_handler>
```

```
     </service_handler_list>
     <terminal_handler>
       <cics_soap_1.2_handler/>
     </terminal_handler>
   </service>
   <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Here is an example SOAP message with an ICRX identity, using blind trust:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <wsse:Security
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
        SOAP-ENV:mustUnderstand="1">

      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
-soap-message-security-1.0#Base64Binary"
        wsu:Id="ICRX"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"
        ValueType="http://www.IBM.com/xmlns/prod/zos/saf#ICRXV1">

        ICRX IS HERE

      </wsse:BinarySecurityToken>

    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>

   APPLICATION SPECIFIC XML IS HERE

  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. Ensure that WebSphere DataPower is configured to be able to send ICRX information. See Sample network topologies for using identity propagation.

Results

Web service requests from WebSphere DataPower with an ICRX identity token in the WS-Security header, connected over a client-certified SSL connection, can now flow.

Related tasks:

Configuring RACF for identity propagation

Configuring IPIC connections for identity propagation

Configuring the pipeline for Web Services Security

To configure a pipeline to support Web Services Security (WSS), you must add a security handler to your pipeline configuration files. You can use the security handler supplied with CICS, as described, or create your own.

Before you begin

Before you define the CICS-supplied security handler, you must identify or create the pipeline configuration files to which you add configuration information for WSS.

Procedure

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.

2. Optional: Code an <authentication> element.
   - In a service requester pipeline, the <authentication> element specifies the type of authentication that must be used in the security header of outbound SOAP messages.
   - In a service provider pipeline, the element specifies whether CICS uses the security tokens in an inbound SOAP message to determine the user ID under which work is 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 265.

   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 265.

   c. In the <authentication> element, code these elements:

      1) An optional, empty <suppress/> element.

         If this element is specified in a service provider pipeline, the handler does not attempt to use any security tokens in the message to determine under which user ID the work runs.

         If this element is specified in a service requester pipeline, the handler does not attempt to add to the outbound SOAP message any of the security tokens that are required for authentication.

      2) In a requester pipeline, an optional <algorithm> element that specifies the URI of the algorithm that is used to sign the body of the SOAP message. You must specify this element if the combination of trust and mode attribute values indicate that the messages are signed. You can specify only the RSA with SHA1 algorithm in this element. The URI is http://www.w3.org/2000/09/xmldsig#rsa-sha1.

      3) An optional <certificate_label> element that specifies the label that is associated with an X.509 digital certificate installed in RACF. If you specify this element 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.

         This element is ignored in a service provider pipeline.

3. Optional: Code an <sts_authentication> element as an alternative to the <authentication> element. You must not code both in your pipeline configuration file. This element specifies that a Security Token Service (STS) is used for authentication and determines the type of request that is sent.

a. Optional: In service provider mode only, code the **action** attribute to specify whether the STS verifies or exchanges a security token. For details of the **action** attribute, see "The <sts_authentication> element" on page 269.

b. Within the <sts_authentication> element, code these elements:

1) An <auth_token_type> element. This element is required when you specify a <sts_authentication> element in a service requester pipeline and is optional in a service provider pipeline. For more information, see <auth_token_type>.

- In a service requester pipeline, the <auth_token_type> element indicates the type of token that STS issues when CICS sends it the user ID contained in the DFHWS-USERID container. The token that CICS receives from the STS is placed in the header of the outbound message.
- In a service provider pipeline, the <auth_token_type> element is used to determine the identity token that CICS takes from the message header and sends to the STS to exchange or validate. CICS uses the first identity token of the specified type in the message header. If you do not specify this element, CICS uses the first identity token that it finds in the message header. CICS does not consider the following as identity tokens:
  - wsu:Timestamp
  - xenc:ReferenceList
  - xenc:EncryptedKey
  - ds:Signature

2) In a service provider pipeline only, an optional, empty <suppress/> element. If this element is specified, the handler does not attempt to use any security tokens in the message to determine the user ID that the work runs under. The <suppress/> element includes the identity token that is returned by the STS.

4. Optional: Code an <sts_endpoint> element. Use this element only if you have also specified an <sts_authentication> element. In the <sts_endpoint> element, code the following elements:

- An <endpoint> element. This element contains a URI that points to the location of the Security Token Service (STS) on the network. It is recommended that you use SSL or TLS to keep the connection to the STS secure, rather than using HTTP.

  To use SAML support, set the endpoint to cics://PROGRAM/DFHSAML.

  You can also specify a WebSphere MQ endpoint, by using the JMS format of URI.
- An optional <jvmserver> element. This element identifies the JVM server that is configured to run the SAML token service. If this element is not included, the default sample resource JVM server DFHXSTS is assumed. This element is valid only if you are using SAML: if you use it in other situations, an error occurs.

5. 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.

6. Optional: If you want to reject inbound SOAP messages that are digitally signed, code an empty <reject_signature/> element.

7. 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.
8. If you want to reject inbound SOAP messages that are partially or fully encrypted, code an empty <reject_encryption/> element.
9. Optional: If you require outbound SOAP messages to be signed, code a <sign_body> element.
   a. In the <sign_body> element, code an <algorithm> element.
   b. Following the <algorithm> element, code a <certificate_label> element.

   Here 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>
```

10. Optional: If you require outbound SOAP messages to be encrypted, code an <encrypt_body> element.
    a. In the <encrypt_body> element, code an <algorithm> element.
    b. Following the <algorithm> element, code a <certificate_label> element.

    Here 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 security handler in which most of the optional elements are present:

```
<wsse_handler>
    <dfhwsse_configuration version="1">
      <authentication trust="signature" mode="basic">
        <suppress/>
        <certificate_label>AUTHCERT03</certificate_label>
      </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>
```

Writing a custom security handler

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.

Before you begin

You need to decide the level of security that your security handler must support, and ensure that an appropriate SOAP fault is returned when a message includes security that is not supported.

About this task

The message handler must also be able to cope with security on inbound and outbound messages.

Procedure

1. Retrieve the DFHREQUEST or DFHRESPONSE container using an **EXEC CICS GET CONTAINER** command.
2. Parse the XML to find the security token that is in the WS-Security message header. The header starts with the `<wsse:Security>` element. The security token might be a user name and password, a digital certificate, or an encryption key. A message can have many tokens in the security header, so your handler needs to identify the correct one to process.
3. Perform the appropriate processing, depending on the security that is implemented in the message.
   a. If you want to perform basic authentication of a Kerberos token, issue an **EXEC CICS VERIFY TOKEN** command. This command checks that the supplied Kerberos token is valid. If the command is successful, update the DFHWS-USERID container with an **EXEC CICS PUT CONTAINER**. Otherwise, issue an **EXEC CICS SOAPFAULT CREATE** command.
   b. If you want to perform basic authentication of a password or password phrase, issue an **EXEC CICS VERIFY PHRASE** command. This command checks the user name and password in the security header of the message. If the command is successful, update the DFHWS-USERID container with an **EXEC CICS PUT CONTAINER**. Otherwise, issue an **EXEC CICS SOAPFAULT CREATE** command.
   c. If you want to perform advanced authentication, either by exchanging or validating a range of tokens with a Security Token Service, use the Trust client interface. See "Invoking the Trust client from a message handler" on page 588 for details.
   d. Validate the credentials of the digital certificate if the message is signed.
   e. If parts of the message are encrypted, decrypt the message using the information in the security header. The Web Services Security: SOAP Message Security specification provides information about how to do this.

Results

Define your security handler program in CICS and update the pipeline configuration file, ensuring that it is correctly placed in the XML. In a service requester pipeline configuration file, the security handler must be configured to run at the end of the pipeline. In a service provider pipeline configuration file, the security handler must be configured to run at the beginning of the pipeline.

What to do next

For general information about how to write a custom message handler, see the IBM Redbooks publication *Application Development for CICS Web Services* that is available from http://www.redbooks.ibm.com/abstracts/sg247126.html.

Invoking the Trust client from a message handler

CICS provides an interface so that you can write your own message handler to invoke a Security Token Service (STS). With this interface you can perform more advanced processing than the CICS-supplied security handler.

Before you begin

About this task

You can use the Trust client instead of the security handler or in addition to it. To use the Trust client interface:

Procedure

1. Extract the correct token from the security message header of the inbound or outbound message.
2. Link to program DFHPIRT, passing the channel DFHWSTC-V1 and the following required containers:
   - DFHWS-STSURI, containing the location of the STS on the network.
   - DFHWS-STSACTION, containing the URI of the type of request that the STS must perform. The two supported actions are `issue` and `validate`.
   - DFHWS-IDTOKEN, containing the token that must either be verified or exchanged by the STS.
   - DFHWS-TOKENTYPE, containing the type of token that the STS must send back in the response.
   - DFHWS-SERVICEURI, containing the URI of the web service operation that is being invoked.

   You can optionally include the DFHWS-XMLNS container to provide the namespaces of the SOAP message that contains the security token. This container is described in more detail in "The header processing program interface" on page 287.
3. DFHPIRT returns with the response from the STS. A successful response is stored in the DFHWS-RESTOKEN container.

   If the STS encounters a problem with the request, it returns a SOAP fault. DFHPIRT puts the SOAP fault in the DFHWS-STSFAULT container. If the STS provides a reason for issuing the SOAP fault, the reason is put in the DFHWS-STSREASON container.

   If an abend occurs, a DFHERROR container is returned that contains details of the processing error.

   Your message handler must handle these responses and perform suitable processing in the event of an error. For example, the message handler might return a suitable SOAP fault to the web service requester.
4. Process the response as appropriate. In provider mode, your pipeline processing must ensure that a user name that CICS can understand is placed in the DFHWS-USERID container by the time the message reaches the application handler. In requester mode, your message handler must add the correct token to the outbound message security header.

What to do next

When you have written your message handler, deploy the program in CICS and update the appropriate pipeline configuration files. In service requester pipelines,

define your message handler to occur at the end of the pipeline processing but before the CICS-supplied security handler. In service provider pipelines, define your message handler at the beginning of the pipeline but after the CICS-supplied security handler.

Related reference:

"DFHWS-STSURI container" on page 315
DFHWS-STSURI is a container of DATATYPE(CHAR). It contains the absolute URI of the Security Token Service (STS) that is used to validate or issue an identity token for the SOAP message.

"DFHWS-STSACTION container" on page 314
DFHWS-STSACTION is a container of DATATYPE(CHAR). It contains the URI of the action that the Security Token Service (STS) takes to either validate or issue a security token.

"DFHWS-IDTOKEN container" on page 313
DFHWS-IDTOKEN is a container of DATATYPE(CHAR). It contains the token that the Security Token Service (STS) either validates or uses to issue an identity token for the message.

"DFHWS-TOKENTYPE container" on page 315
DFHWS-TOKENTYPE is a container of DATATYPE(CHAR). It contains the URI of the requested token type that the Security Token Service (STS) issues as an identity token for the SOAP message.

"DFHWS-SERVICEURI container" on page 314
DFHWS-SERVICEURI is a container of DATATYPE(CHAR). It contains the URI that the Security Token Service (STS) uses as the AppliesTo scope.

"DFHWS-RESTOKEN container" on page 313
DFHWS-RESTOKEN is a container of DATATYPE(CHAR). It contains the response from the Security Token Service (STS).

"DFHWS-STSFAULT container" on page 314
DFHWS-STSFAULT is a container of DATATYPE(CHAR). It contains the error that was returned by the Security Token Service (STS).

"DFHWS-STSREASON container" on page 314
DFHWS-STSREASON is a container of DATATYPE(CHAR). It contains the contents of the `<wst:Reason>` element, if this element is present in the response message from the Security Token Service (STS).

"DFHERROR container" on page 291
DFHERROR is a container of DATATYPE(BIT) that is used to convey information about pipeline errors to other message handlers.

Chapter 14. Interoperability between the web services assistant and WSRR

The CICS web services assistant can interoperate with the IBM WebSphere Service Registry and Repository (WSRR). Use WSRR to find web services that you are requesting more quickly and enforce version control of the web services that you are providing.

Both DFHLS2WS and DFHWS2LS include parameters to interoperate with WSRR. DFHLS2WS also includes an optional parameter so that you can add your own customized metadata to the WSDL document in WSRR.

If you want the web services assistant to communicate securely with WSRR, you can use secure socket level (SSL) encryption. Both DFHLS2WS and DFHWS2LS include parameters for using SSL encryption.

To use SSL with the web services assistant and WSRR, see "Example of how to use SSL with the web services assistant and WSRR" on page 206.

Example of how to use SSL with the web services assistant and WSRR

You can interoperate securely between the web services assistant and an IBM WebSphere Service Registry and Repository (WSRR) server by using secure socket layer (SSL) encryption. To use SSL encryption you need a key store and a trust store; you must also specify certain parameters on the web services assistant.

About this task

Complete the following steps to use SSL encryption for interactions between the web services assistant and WSRR.

Procedure

1. Create a key store for your private keys and public key certificates (PKC).
   a. You can create a key store using a key configuration program such as the IBM Key Management Utility (iKeyman).
   b. Specify the **SSL-KEYSTORE** parameter in DFHWS2LS or DFHLS2WS with the fully qualified name of the key store that you have created.
   c. Optional: Specify the **SSL-KEYPWD** parameter in DFHWS2LS or DFHLS2WS with the password of the key store that you have created.
2. Create a trust store for all your trusted root certificate authority (CA) certificates. These certificates are used to establish the trust of any inbound public key certificates.
   a. You can create a trust store using a key configuration program such as the IBM Key Management Utility (iKeyman).
   b. Specify the **SSL-TRUSTSTORE** parameter in DFHWS2LS or DFHLS2WS with the fully qualified name of the trust store that you have created.
   c. Optional: Specify the **SSL-TRUSTPWD** parameter in DFHWS2LS or DFHLS2WS with the password of the trust store that you have created.
3. Test that the web services assistant is able to communicate with WSRR using SSL encryption.

a. You can use the sample files provided by IBM WebSphere Application Server to test the web services assistant with WSRR.
   - The sample key stores provided by WebSphere Application Server are `DummyClientKeyFile.jks` and `DummyServerKeyFile.jks`.
   - The sample trust stores provided by WebSphere Application Server are `DummyClientTrustFile.jks` and `DummyServerTrustFile.jks`.

b. Replace the keys in the sample key and trust store files. These keys are shipped with WebSphere Application Server and must be replaced for security.

Results

The web services assistant can now use SSL encryption to communicate securely with WSRR across a network.

Chapter 15. Troubleshooting SOAP web services

The problems that you might get when implementing SOAP 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 CICS web services assistant batch jobs or the CICS XML assistant batch jobs, install a PIPELINE resource in CICS, or install a WEBSERVICE resource in CICS. The most common deployment errors are described here, including the symptom of the problem, the cause and the solution.

About this task

In the event of a deployment error, PIPELINE resources typically install in a DISABLED state and WEBSERVICE resources install in an UNUSABLE state.

Information and error messages associated with the CICS web services assistant batch jobs and the CICS XML assistant batch jobs are located in the job log. Error messages associated with installing resources are located in the system log.

Codes of 0, 4, 8, or 12 are issued by the assistants, other codes are typically issued by BPXBATCH, the JVM, or IEBGENER.

Codes issued by BPXBATCH fall into two main categories: a code of less than 128 indicates a command failure, a code of greater than or equal to 128 indicates that the process was terminated by a signal. For more information about BPXBATCH and its return codes, see the *z/OS UNIX System Services Command Reference*.

Procedure

- You receive a return code of 0, 4, 8, or 12 when running the CICS web services assistant batch jobs or the CICS XML assistant batch jobs. The return codes mean the following:
  - 0 - The job completed successfully.
  - 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 should 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 return code of 1, 136 or 139 when running the CICS web services assistant batch jobs or the CICS XML assistant batch jobs. These return codes indicate that the JVM has failed, usually because there is insufficient storage available. The CICS assistants require a JCL region size of at least 200 MB.
  1. Increase the region size, or consider setting the region size to `0M`.
  2. Check for any active IEFUSI exits, which can limit the region size.
- You receive a return code of 137 when running the CICS web services assistant batch job DFHLS2WS, or the CICS XML assistant batch job DFHLS2SC. This return code means that the job timed out.
  1. Increase the time by coding the **TIME** parameter on the EXEC statement of your job to **TIME=1440**, or increase the MAXCPUTIME value in the SYS1.PARMLIB(BPXPRMxx) member.
- You receive a DFHPI0914 error message when attempting to install a WEBSERVICE resource. The message includes some information about the cause of the installation 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 the *CICS Transaction Server for z/OS Installation Guide* 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.
- The WEBSERVICE resource is in a DISABLED state. The states DISABLED and DISABLING are only available for WEBSERVICE resources that are defined and installed in CICS bundles.
  1. If the PIPELINE resource associated with the WEBSERVICE resource has been discarded, the WEBSERVICE resource enters DISABLED state. Investigate why the PIPELINE resource is missing, and replace it if appropriate.
  2. If a disable action has been carried out for the CICS bundle where the WEBSERVICE resource is defined, the WEBSERVICE resource enters DISABLED state when the web service is no longer in use. Investigate the state of the CICS bundle, and enable it if appropriate.

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.

Procedure

- You receive a DFHPI0401, DFHPI0502, or similar message, indicating that an HTTP or WebSphere MQ transport error has occurred. If the transport is HTTP, the client receives a 500 Server Internal Error message. If the transport is WebSphere MQ, the message is written to the dead letter queue (DLQ). A SOAP fault is not returned to the web service requester, because CICS cannot determine what type of message was received.
- You receive a DFH*xx* 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 is in an UNUSABLE state, see "Diagnosing deployment errors" on page 593.
  3. Check that you have correctly specified the URI and port number. In particular, check the case, because 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 indicate 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 WebSphere MQ 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, 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, because 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.For more information about CEDX, see Using the CEDX transaction in *CICS Supplied Transactions*.
3. If CEDX does not 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 601.
- If you think your problem is related to MTOM messages, see "Diagnosing MTOM/XOP errors" on page 598.
- If a persistent HTTP connection is periodically closed:
  1. Check whether performance tuning for HTTP connections is enabled. For more information, see SOTUNING
  2. If performance tuning is enabled, CICS will periodically close persistent HTTP connections to allow connections to be redistributed among regions that are listening on shared IP endpoints.
- If connection attempts are refused when your CICS region is at maximum capacity.
  1. Check whether performance tuning for HTTP connections is enabled. For more information, see SOTUNING
  2. If performance tuning is enabled, when CICS is at maximum capacity all inbound HTTP connection open requests will queue outside of CICS in the TCPIPSERVICE's listening connection's backlog queue in TCP/IP. The region's TCP/IP Global Statistics SOG_PAUSING_HTTP_LISTENING field reflects whether it is currently the case, and the SOG_TIMES_AT_ACCEPT_LIMIT/ SOG_TIME_LAST_PAUSED_HTTP_LISTENING fields contain more information on past occurrences.
  3. Use **NETSTAT ALL** to obtain ConnectionsDropped for the TCPIPSERVICE's listening connection. This is the number of connection requests received and dropped because the maximum number of connection requests was already in the backlog queue. For more information, see Netstat in the z/OS Communication Server IP System Administrator's Commands.
  4. If the ConnectionsDropped value is too high, consider increasing the TCPIPSERVICE's backlog attribute's value.

Diagnosing service requester runtime errors

Read this section 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.

About this task

Problems that occur can be due to errors in individual web services or issues at the transport level.

Procedure

- If you are using the **INVOKE SERVICE** 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 SERVICE 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 287 for details.
  3. If the web service requester application is not using the **INVOKE SERVICE** command to send a web service request, check that it has created all of the necessary control containers and that they are the correct 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 SERVICE** command and a RESP value of 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 601.
  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 SERVICE** 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 usually returned as a SOAP fault. If your program is using the

INVOKE SERVICE command, CICS returns a RESP value of TIMEDOUT and RESP2 code of 2 for a timeout error, and a RESP value of INVREQ and RESP2 code of 17 for a transport error.

1. Check that the network end point is present.
2. Ensure that the RESPWAIT attribute on the PIPELINE resource is correctly configured to meet your application's requirements. The RESPWAIT attribute defines how long CICS waits for a reply from the web service provider before returning to the application. If no value is specified, CICS uses the defaults of 10 seconds for HTTP and 60 seconds for WebSphere MQ. However, CICS also has a time out in the dispatcher for each transaction, and if this is less than the default of the protocol that is being used, CICS uses the dispatcher time out instead.

- 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.
- If you think the problem is related to sending or receiving MTOM messages, see "Diagnosing MTOM/XOP errors."

Diagnosing MTOM/XOP errors

MTOM/XOP errors can occur at run time, in both requester and provider mode pipelines.

Before you begin

If you are having problems configuring a pipeline to support MTOM/XOP, read "Diagnosing deployment errors" on page 593.

About this task

Procedure

- If you are able to send a web service request message in MTOM format, but are getting a SOAP fault message from the web service provider, look at the contents of the DFHWS-BODY container for details of the SOAP fault.
  1. Is the web service provider able to receive MIME Multipart/Related messages? If the web service provider does not support the MTOM format, the fault that you get back can vary depending on the implementation. If the web service provider is another CICS application, the SOAP fault would indicate that the MIME message is not a valid content type.

2. If the web service provider can receive MIME messages, check to see if the pipeline is sending the message in direct or compatibility mode. If you are sending an MTOM message in direct mode, there could be a problem with the XML.
3. To find out if the problem is with the XML, turn validation on for the web service. This causes the MTOM message to be processed in compatibility mode through the pipeline. As part of this processing, the MTOM handler parses the message contents to optimize the base64binary data. If there is an error in the XML, CICS puts the error in the DFHERROR container and issues an MTOM transport failure in the pipeline.
4. Examine the contents of the DFHERROR container to see if this indicates what problem occurred. If this isn't enough information to help you diagnose the cause of the problem, run a level 2 trace of the PI domain.
5. Look for trace point PI 0C16. This describes the problem that was encountered in more detail, and should help you to fix the problem with the XML that is provided by the requester application.

- If expected binary attachments are missing from the outbound MTOM message, this could indicate that the binary data is considered too small to optimize as a binary attachment. CICS only creates binary attachments for data that is large enough to justify the processing overhead of optimizing it in the pipeline. Any binary data below 1,500 bytes in size is not optimized.
- If you are unable to send an outbound MTOM message in compatibility mode and the pipeline is returning a DFHERROR container, there was a problem when the pipeline tried to process the MTOM 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. Check that the XML in your outbound MTOM message is valid. CICS does not perform any validation on outbound messages in the pipeline.
- If you receive a DFHPI1100E message, there was a problem with the MIME headers of an MTOM message that was received by CICS. The CICS message contains the general class of MIME error that occurred. To find the exact problem that occurred:
  1. If you have auxiliary trace active in your CICS region, check for any exception trace entries.
  2. Look for trace point PI 1305. This describes the nature of the MIME header error, the location of the error in the header, and up to 80 bytes of text before and after the error so you can understand the context of where the error occurred.

  For example, the following excerpt of trace indicates that the MIME content-type start parameter was invalid because it was not enclosed in quotes, but included characters that are not valid outside a quoted string.

```
PI 1305 PIMM *EXC* - MIME_PARSE_ERROR -

            TASK-01151 KE_NUM-0214 TCB-QR   /009C7B68 RET-9C42790A TIME-10:33:41.3667303015 INTERVAL-00.0000053281        =000599=
              1-0000  C5A79785 83A38584 40978199 819485A3  859940A5 8193A485 40A39692 85954096  *Expected parameter value token o*
                0020  994098A4 96A38584 40A2A399 899587                                      *r quoted string                 *
              2-0000  D4C9D4C5 40A2A895 A381A740 85999996  994081A3 404EF0F0 F0F0F1F1 F2408995  *MIME syntax error at +0000112 in*
                0020  40C39695 A38595A3 60A3A897 85408885  81848599                            * Content-type header            *
              3-0000  5F626F75 6E646172 793B2074 7970653D  22617070 6C696361 74696F6E 2F786F70  *_boundary; type="application/xop*
                0020  2B786D6C 223B2073 74617274 2D696E66  6F3D2261 70706C69 63617469 6F6E2F73  *+xml"; start-info="application/s*
                0040  6F61702B 786D6C22 3B207374 6172743D                                      *oap+xml"; start=                *
              4-0000  3C736F61 70736C75 6E674074 6573742E  68757273 6C65792E 69626D2E 636F6D3E  *<soapslung@test.hursley.ibm.com>*
                0020  3B206368 61727365 743D7574 662D38                                      *; charset=utf-8                 *
```

- The pipeline processing fails to parse an inbound MTOM message, and the web service requester receives a SOAP fault message. This indicates that there was a problem with the XOP document in the MTOM message. In direct mode, the SOAP fault is generated by the application handler. If the pipeline is running in

compatibility mode, the message is parsed by the MTOM handler when constructing the SOAP message. In this case, CICS issues a DFHPI prefixed error message and a SOAP fault.

1. The DFHPI prefixed error message indicates what was wrong with the XOP document. For example, it could be an invalid MIME header or a missing binary attachment.
2. To find the exact cause of the problem, check for any exception trace points. In particular, look for trace points beginning with PI 13*xx*. This describes the exception that occurred in more detail.

You can also run a PI level 2 trace to establish the sequence of events leading up to the error, but this can have a significant performance impact and is not recommended on production regions.

Determining if a web service supports MTOM/XOP

You can find out whether a web service supports MTOM/XOP with the following interfaces:

CICS Explorer

The CICS Explorer administration views

Use the **XOP Direct Status** and **XOP Support Status** attributes in the Web Services view.

CICSPlex SM

The WEBSERVICE definitions view

CEMT

The INQUIRE WEBSERVICE command

The CICS SPI

The INQUIRE WEBSERVICE command

Checking the operation mode of a PIPELINE resource

You can check the operation mode of a PIPELINE with the following interfaces:

CICS Explorer

The CICS Explorer administration views

Use the **Operation mode** attribute in the Pipelines view.

CICSPlex SM

The PIPELINE definitions view

CEMT

The INQUIRE PIPELINE command

The CICS SPI

The INQUIRE PIPELINE command

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.

Before you begin

Symptoms include the generation of SOAP fault messages and CICS messages indicating that a failure has occurred.

About this task

If you have a data conversion problem, perform the following steps:

Procedure

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 can 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 off validation: `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 an inbound SOAP message 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 occur, they are followed by a conversion error. For a possible interpretation of these conversion errors, see the explanations of messages DFHPI1007 to DFHPI1010.

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.

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 SERVICE** command returns a RESP code of INVREQ and a RESP2 value of 14.

Chapter 16. The CICS catalog manager example application

The CICS catalog manager 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 a user can perform these tasks:

- 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.

For this example, the CICS Explorer is used to install and deploy the application. The CICS Explorer is an Eclipse-based graphical tooling interface for CICS.

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 40 on page 606 shows the structure of the base application.

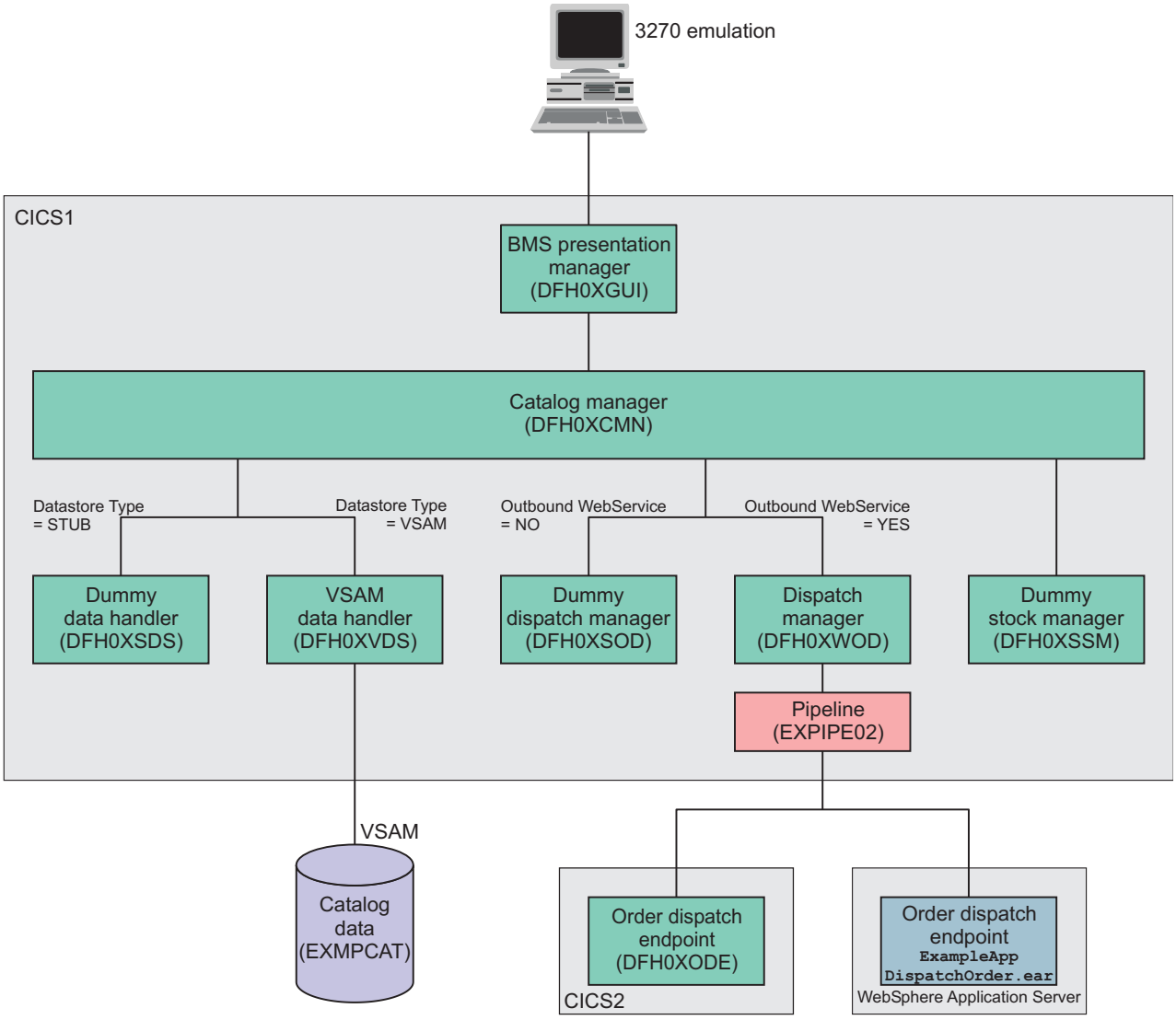


Figure 40. Structure of the base application

The components of the base application are:

- A BMS presentation manager (DFH0XGUI) that supports a 3270 terminal or emulator, and that interacts with the main catalog manager program.
- A catalog manager program (DFH0XCMN) that is the core of the example application, and that interacts with several back-end components:
  - 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 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: DFHX0WOD is a web service requester that invokes a remote order dispatch end point, and DFHX0SOD is a dummy program that returns valid responses to its caller.

There are two equivalent order dispatch endpoints: DFH0XODE is a CICS service provider program; `DispatchOrderV7.war` is a Java web archive file that can be deployed in CICS Liberty JVM Server or similar environments.

– A dummy stock manager program (DFH0XSSM) 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 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 when 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 program

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 create two TRANSACTION resources.

Creating and defining the VSAM data sets

Two KSDS VSAM data sets are used to define and populate the example application. One data set contains configuration information for the example application. The other contains the sales catalog.

Procedure

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 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.

      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.
3. Run both jobs to create and populate the data sets.
4. Use the CICS Explorer to create a FILE definition for the catalog file.
   a. Select **Definitions** > **File Definitions**. Right-click in the **Name** column and click **New** to create a new file definition. Type EXAMPLE in the **Resource Group** text box, and type EXMPCAT in the **Name** text box. Click **Finish** to

define the FILE definition. Alternatively, you can copy the file definition from the CICS supplied group DFH$EXBS.

b. Double-click the new EXMPCAT file. In the File Definition (EXMPCAT) CICS Example Application editor, select the **VSAM** tab. Type *hlq*.EXMPLAPP.EXMPCAT in the **Data set name to be used** text box.

c. Select the **Attributes** tab and set the operations of the following attributes to **Yes**:

   Add
   Browse
   Delete
   Read
   Update

d. Use the default values for all other attributes.

5. Use the CICS Explorer to create a FILE definition for the configuration file.

a. Select **Definitions** > **File Definitions**. Right-click in the **Name** column and click **New** to create a new file definition. Type EXAMPLE in the **Resource Group** text box, and type EXMPCONF in the **Name** text box. Click **Finish** to define the FILE definition. Alternatively, you can copy the file definition from the CICS supplied group DFH$EXBS.

b. Double-click the new EXMPCONF file. In the File Definition (EXMPCONF) CICS Example Application window, select the **VSAM** tab. Type *hlq*.EXMPLAPP.EXMPCONF in the **Data set name to be used** text box.

c. Select the **Attributes** tab and set the operations of the following attributes to **Yes**:

   Add
   Browse
   Delete
   Read
   Update

d. Use the default values for all other attributes.

Results

The data sets are populated, and the FILE definitions for the catalog file and the configuration file have been created and are ready to install.

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.

Procedure

1. Create TRANSACTION definitions for the following transactions using the CICS Explorer. The correct operation of the example application does not depend on the names of the transactions, so you can use different names.

a. Copy the definitions for transaction EGUI from the CICS supplied group DFH$EXBS by right-clicking DFH$EXBS in the Resource Group Definitions view. Select **New** > **Transaction Definition**. Type EGUI in the **Name** text box, and DFH0XGUI in the **Program Name** text box. Click **Finish** to create the EGUI TRANSACTION definition.

b. Copy the definitions for transaction ECFG from the CICS supplied group DFH$EXBS by right-clicking DFH$EXBS in the Resource Group Definitions window. Select **New** > **Transaction Definition**. Type `ECFG` in the **Name** text box, and `DFH0XCFG` in the **Program Name** text box. Click **Finish** to create the ECFG TRANSACTION definition.

c. Optional: Copy the definitions for transactions ECLI from the CICS supplied group DFH$EXWS by right-clicking DFH$EXWS in the Resource Group Definitions view. Select **New** > **Transaction Definition**. Type `ECLI` in the **Name** text box, and `DFH0XCUI` in the **Program Name** text box. Click **Finish** to create the ECLI transaction definition.

Use the default values for all other attributes.

2. Optional: If you do not want to use program autoinstall, copy the PROGRAM definitions for the base application programs and the MAPSET definitions for the BMS maps from the CICS supplied group DFH$EXBS.

   a. Copy the 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 636.

   b. Copy the PROGRAM resource definitions for the following COBOL programs.

Table 21. 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 the controller program.
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. This program 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. This program sets the text `"Order in dispatch"` in the return COMMAREA.
DFH0XSDS	A *stubbed* or dummy version of the data store program that allows the application to work when the VSAM catalog file has not been set up. DFH0XSDS 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. DFH0XSOD is used when outbound web services are not required.
DFH0XSSM	A stubbed version of the stock manager (replenishment) program. DFH0XSSM sets the return code in the COMMAREA to 0 and returns to its caller.
DFH0XVDS	The VSAM version of the data store program. DFH0XVDS accesses the VSAM file to perform reads and updates of the catalog.
DFH0XWOD	The web service version of the order dispatch program. DFH0XWOD 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: Copy the PROGRAM definitions for DFH0XCUI from the CICS supplied group DFH$EXWS. 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.

```
DIS G(DFH$EXWS)
ENTER COMMANDS
NAME     TYPE          GROUP
DFH0XCUI PROGRAM       DFH£EXWS
ECLI     TRANSACTION   DFH£EXWS
EXMPPORT TCPIPSERVICE  DFH£EXWS
EXPIPE01 PIPELINE      DFH£EXWS
EXPIPE02 PIPELINE      DFH£EXWS
```

Completing the installation

To complete the installation, install the RDO group that contains your resource definitions.

Procedure

Right-click the resource group in the Resource Group Definitions window. Select **Install**. Make sure that your CICSplex is correct and that you select your target region, then click **OK**.

Results

Your RDO is now installed and the application is ready for use.

Configuring the example application

The base application includes a transaction (ECFG) that you can use to configure the example application.

Before you begin

The configuration transaction uses mixed-case information. You must use a terminal that can handle mixed-case information correctly.

About this task

You can 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

With the configuration transaction, you can replace CICS-supplied components of the example application with your own without restarting the application.

Procedure

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 the **Outbound WebService** to: http://*myserver*:*myport*/exampleApp/dispatchOrder where *myserver* and *myport* are your CICS server address and port.

Running the example application with the BMS interface

The base application can be run using the BMS interface.

Procedure

1. Enter transaction EGUI from a CICS terminal. The example application menu is displayed:

```
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
```

   You can list the items in the catalog, order an item, or exit the application using the options on the menu.
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 20lb 500/ream                 5.35     _
0100    Green Laser Paper 20lb 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 fulfill the order, enter the following information:
   a. Complete the **Order 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 **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. Press F3 to end the applications.

Web service support for the example application

The web service support extends the example application, providing two versions of web client front end and two versions of a web service endpoint for the order dispatcher component.

The two versions of web client front end and one version of the web service endpoint are supplied as Java web archive files (WARs) that run in the following environments:

Environment	WAR files

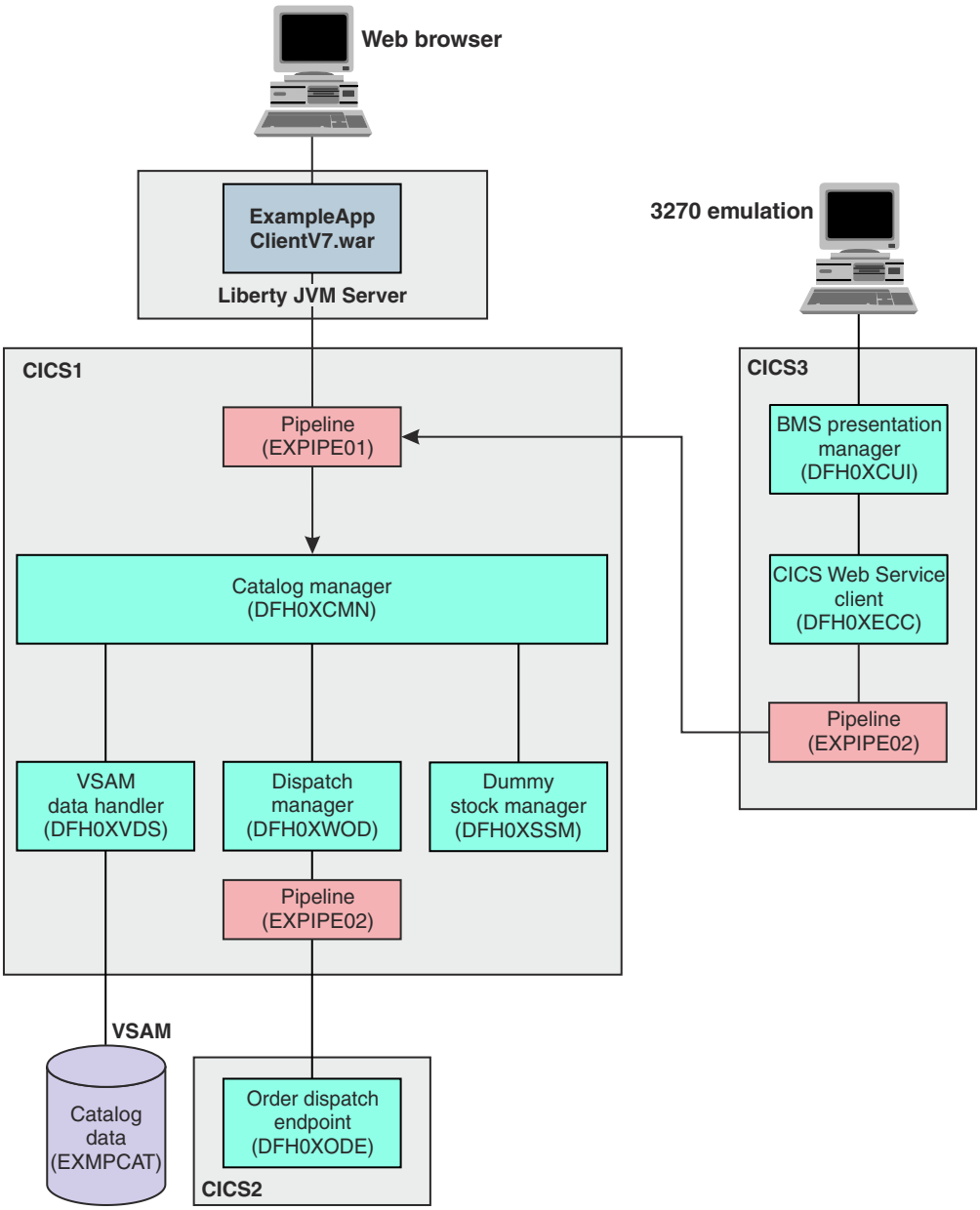
CICS Liberty JVM Server V8.5.5	`ExampleAppClientV7.war`
	`ExampleAppWrapperClientV7.war`
	`DispatchOrderV7.war`

These WARs were exported from dynamic web projects. For deploying the WAR files please see,

Please note that you will need to enable the jax-ws feature in the Liberty JVM server by adding, e.g. `</feature> jaxws-2.2 </feature>` in JVM server's configuration file e.g. `server.xml.`

The second version of the web service endpoint is supplied as a CICS service provider application program (DFH0XODE).

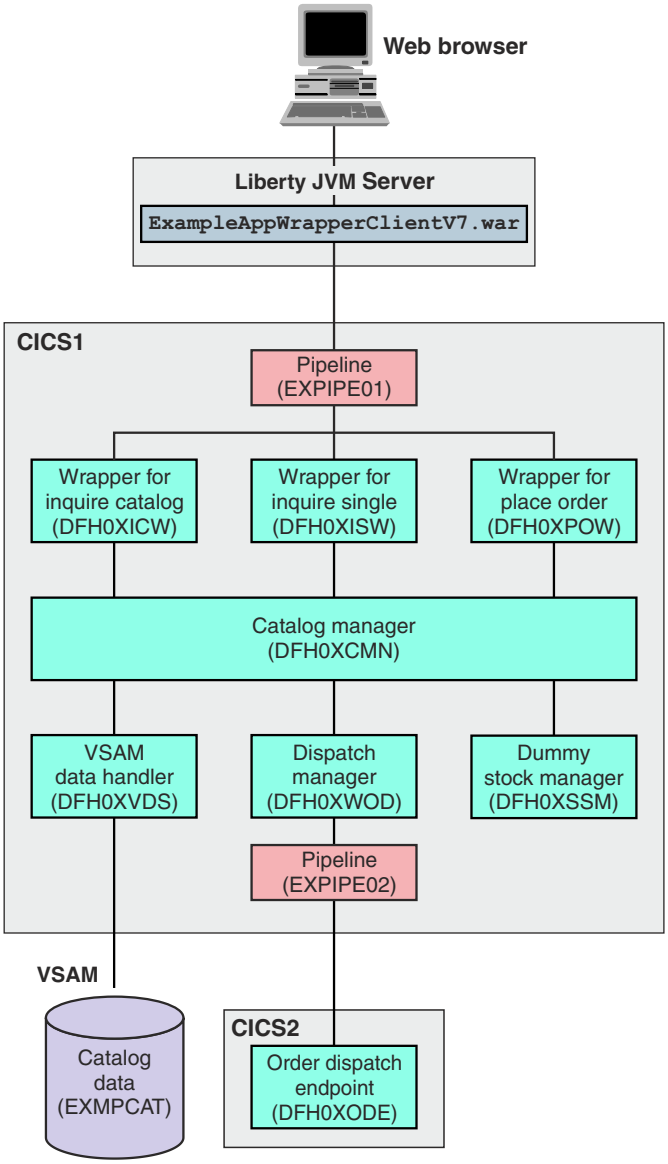
Figure 1 shows a configuration of the example application with one version of the web client front end and CICS service provider as the order dispatch web service end point. It also includes a web service client on a CICS system.



In this configuration, the application is accessed through two different clients:

- A web browser client connected to Liberty JVM Server, in which `ExampleAppClientV7.War` 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 2 shows another version of web client front end, with CICS service provider as the order dispatch web service end point.

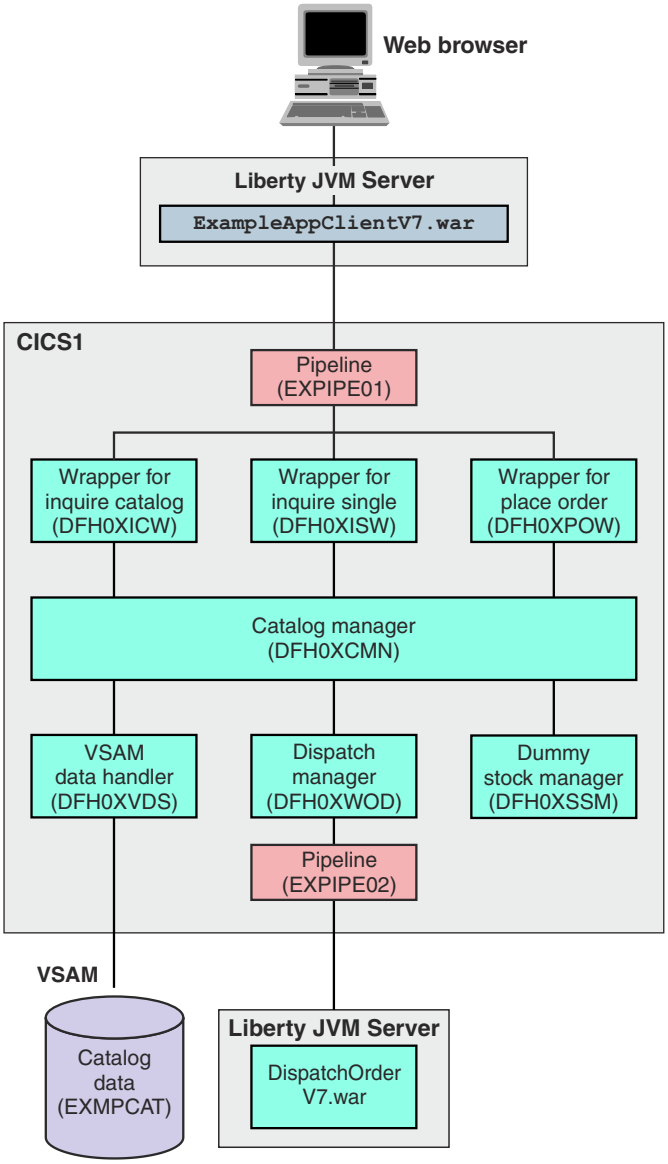


In this configuration, the web browser client is connected to Liberty JVM Server, in which `ExampleAppWrapperClientV7.war` is deployed. In CICS, three wrapper applications (for the inquire catalog, inquire single, and place order functions) are deployed as service provider applications. They in turn link to the base application.

In order for Dispatch Manager on your CICS system to call this end point, you need to change the following configuration:

- Outbound WebService? to YES
- Outbound WebService URL to the URL where the Dispatch Order end point is being deployed on, e.g. `http://cics2:8080/exampleApp/dispatchOrder`

Figure 3 shows a configuration of the example application with both of the web client front end and the order dispatch web service end point on Liberty JVM



server.

In this configuration, the web browser client is connected to Liberty JVM Server, in which ExampleAppClientV7.war is deployed. The order dispatch web service end point DispatchOrderV7.war is installed on Liberty JVM Server.

In order for Dispatch Manager on your CICS system to call this end point, you need to change the following configuration. Please see Configuring the example application on how to do the configuration

- Outbound WebService? to YES
- Outbound WebService URL to the URL where the Dispatch Order end point is being deployed on, e.g. `http://mylibertyserver:9080/DispatchOrderV7/DispatchOrder`

Please see Configuring the example application on how to do the configuration, see

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.

About this task

The coded character sets used in the example application are:

037 EBCDIC Group 1: USA, Canada (z/OS), Netherlands, Portugal, Brazil, Australia, New Zealand)

1208 UTF-8 Level 3

Procedure

Add the following statements to the conversion image for your z/OS system:

```
CONVERSION 037,1208;
CONVERSION 1208,037;
```

For more information, see the *CICS Transaction Server for z/OS Installation Guide*.

Defining the web service client and wrapper programs

If you are not using program autoinstall, you must define resource definitions for the web service client and wrapper programs.

Procedure

1. Use the CICS Explorer to define PROGRAM resource definitions for the wrapper programs, by selecting **Definitions** > **Program Definitions**. Right-click in the Program Definitions view and select **New** to create a new program definition. Type `DFH0XECC` in the **Resource Group** text box, and type `PROGRAM1` in the **Name** text box. Click **Finish** to define the PROGRAM definition. Create definitions for the following COBOL programs:

Table 22. 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. Use the CICS Explorer to define PROGRAM resource definitions for the web services client program DFH0XECC, by selecting **Definitions** > **Program Definitions**. Right-click in the Program Definitions view and select **New** to create a new program definition. Type `DFH0XECC` in the **Resource Group** text box, and type `PROGRAM1` in the **Name** text box. Click **Finish** to define the program definition. The web services client program DFH0XECC 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 z/OS UNIX directories, and create the required CICS resources.

The z/OS UNIX directories

Web service support for the example application requires a shelf directory and a pickup directory in z/OS UNIX.

The shelf directory is used to store the WEBSERVICE 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 WEBSERVICE 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.

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 act on web service requests and responses as they pass through the pipeline.

About this task

The example application uses the 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 web service 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 because a single pipeline cannot be configured to be both a provider and requester pipeline at the same time.

If you want to use Java-based pipelines, you must specify the sample service provider configuration file `basicsoap11javaprovider.xml` instead of `basicsoap11provider.xml` in step 1b. And specify the sample service requester configuration file `basicsoap11javarequester.xml` instead of `basicsoap11requester.xml` in step 2b. For more information about sample configuration files, see Pipeline configuration files. Also, if you want to use the Axis2 application handler in your Java-based pipeline, you must replace `EXPIPE01` with `EXPIPE03` in step 1a and `EXPIPE02` with `EXPIPE04` in step 2a

Procedure

1. Use the CICS Explorer to create a pipeline definition for the service provider.
   a. Create a PIPELINE definition for the wrapper programs using the CICS Explorer by selecting **Definitions** > **Pipeline Definitions**. Right-click in the Pipeline Definitions view and select **New** to create a new pipeline definition. Type `DFH$EXWS` in the **Resource Group** text box, and type `EXPIPE01` in the **Name** text box. Click **Finish** to create the PIPELINE definition. Alternatively, you can copy the PIPELINE definition from CICS

supplied group DFH$EXWS. Right-click DFH$EXWS in the Resource Group Definition view and select **New** > **Pipeline Definition**.

b. Double-click the PIPELINE definition and select the **Attributes** tab from the Pipeline Definition (EXPIPE01) editor. Under **Details**, the **Configuration File** must be set to the location of the sample files */usr/lpp/cicsts/samples/pipelines/basicsoap11provider.xml*, where */usr/lpp/cicsts* is the path to the files on your directory, **Shelf** must be */var/cicsts/*, **Status** must be ENABLED, and **WS Directory** must be */usr/lpp/cicsts/samples/webservices/wsbind/provider/*.

The z/OS UNIX entries are case-sensitive and assume a default CICS z/OS UNIX installation root of */usr/lpp/cicsts*.

2. Use the CICS Explorer to create a PIPELINE definition for the service requester.

a. Create a PIPELINE definition for the wrapper programs using the CICS Explorer by selecting **Definitions** > **Pipeline Definitions**. Right-click in the Pipeline Definitions view and select **New** to create a new pipeline definition. Type DFH$EXWS in the **Resource Group** text box, and type EXPIPE02 in the **Name** text box. Click **Finish** to create the PIPELINE definition. Alternatively, you can copy the PIPELINE definition from CICS supplied group DFH$EXWS.

b. Double click the PIPELINE definition and select the **Attributes** tab from the Pipeline Definition (EXPIPE02) editor. Under **Details**, **Configuration File** must be set to the location of the sample files */usr/lpp/cicsts/samples/pipelines/basicsoap11requester.xml*, where */usr/lpp/cicsts* is the path to the files on your directory, **Shelf** must be */var/cicsts/*, **Status** must be ENABLED, and **WS Directory** must be */usr/lpp/cicsts/samples/webservices/wsbind/requester/*.

Creating a TCP/IP service

Because the client connects to your web services over an HTTP transport you must define a TCP/IP service to receive the inbound HTTP traffic.

Procedure

Use the CICS Explorer to create a TCPIPSERVICE definition to handle inbound HTTP requests.

1. Create a TCPIPSERVICE definition by selecting **Definitions** > **TCP/IP Service Definitions**. Right-click in the TCP/IP Service Definitions view and select **New** to create a new definition. Type DFH$EXWS in the **Resource Group** text box, and type EXMPPORT in the **Name** text box. You must specify a port number; type in the number of any unused port in your CICS system. Click **Finish** to create the TCPIPSERVICE definition.
2. Double-click the TCPIPSERVICE definition. In the **Attributes** tab in the TCP/IP Service Definition (EXMPPORT) editor, set the following attributes:

   Urm must be DFHWBAAX

   Protocol must be HTTP

   Transaction must be CWXN
3. Use the default values for all other attributes.

Dynamically installing the WEBSERVICE and URIMAP resources

Each function that is 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 web service. Although you can use resource definition

online (RDO) to define and install your WEBSERVICE and URIMAP resources, you can also have CICS create them dynamically when you install a pipeline resource.

Procedure

1. Use the CICS Explorer to install the PIPELINE resources.
   a. Select **Definitions** > **Pipeline Definitions**. Right-click the EXPIPE01 PIPELINE definition in the Pipeline Definitions view and select **Install**. Select your target CICS region by selecting the check box. Click **OK** to install the PIPELINE.

      Note: If you created Java-based pipeline definitions in "Creating the pipeline definition" on page 620 then right-click EXPIPE03 PIPELINE definition in the Pipeline Definitions view.
   b. Repeat this process for the EXPIPE02 PIPELINE definition or EXPIPE04 for Java-based pipelines.

      When you install each PIPELINE resource, CICS scans the directory specified in the PIPELINE WSDIR attribute (the pickup directory). For each WEBSERVICE binding file in the directory, that is for each file with the `.wsbind` suffix, CICS installs a WEBSERVICE resource and one URIMAP resource if these resources do not exist.

      The URIMAP resource provides CICS with the information to associate the WEBSERVICE resource with a specific URI. Existing resources are replaced if the information in the binding file is newer than the existing resources.

      A second optional URIMAP resource is installed if a WSDL file or WSDL archive file has been copied to the pickup directory. This URIMAP resource provides CICS with the information to associate the WSDL archive file or WSDL document with a specific URI so that external requesters can use the URI to discover the WSDL archive file or WSDL document.

      When the PIPELINE is later disabled and discarded, all associated WEBSERVICE and URIMAP resources are also discarded.
2. If you have already installed the PIPELINE resource, use the **PERFORM PIPELINE SCAN** command to initiate the scan of the PIPELINE pickup directory.

   When you have installed the PIPELINE resources, the following WEBSERVICE resources and the associated URIMAP resources for the provider pipeline are installed in your system:

   ```
   dispatchOrder
   dispatchOrderEndpoint
   inquireCatalog
   inquireCatalogClient
   inquireCatalogWrapper
   inquireSingle
   inquireSingleClient
   inquireSingleWrapper
   placeOrder
   placeOrderClient
   placeOrderWrapper
   ```

   The names of the WEBSERVICE resources are derived from the names of the WEBSERVICE binding files; the names of the URIMAP resources are generated dynamically. An additional URIMAP is generated for each WSDL document that exists in the pickup directory of the pipeline. You can view the resources

by selecting **Operations** > **Web Services** to open the Web Services view. Right-click the WEBSERVICE resource and select **Open Related** > **URI Map**.

The CICS Explorer view shows the names of the PIPELINE resource, the URIMAP resource, and the target program that is associated with each web service. In this example, there is no URIMAP or target program for WEBSERVICE(dispatchOrder) because the WEBSERVICE resource is for an outbound request.

WEBSERVICE(dispatchOrderEndpoint) represents the local CICS implementation of the dispatch order service.

Creating the WEBSERVICE resources with resource definition online (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).

Before you begin

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 pickup directory of the PIPELINE. This ensures that the WEBSERVICE and URIMAP resources are not dynamically installed during a pipeline scan of the pickup directory. Alternatively, you can ensure that no value is specified for WSDIR in the PIPELINE. However, if you do not specify a value for WSDIR, no pipeline scans of the pickup directory occur. Therefore, all WEBSERVICE and URIMAP resources have to be created and installed using RDO.

Procedure

1. Use the CICS Explorer to create a WEBSERVICE definition for the inquire catalog function of the example application.
   a. Create a WEBSERVICE definition using the CICS Explorer by selecting **Definitions** > **Web Service Definition**.
   b. Right-click in the Web Service Definitions view and select **New** to create a new WEBSERVICE definition.
   c. Type `DFH$EXWS` in the **Resource Group** text box, type `EXINQCWS` in the **Name** text box, and type `EXPIPE01` in the **Pipeline** text box or type `EXPIPE03` for Java-based pipelines. You must enter the WSBind attribute before you can create the WEBSERVICE definition. In the **WSBind File** text box type `/usr/lpp/cicsts/samples/webservices/wsbind/provider/inquireCatalog.wsbind`.
   d. Click **Finish** to create the WEBSERVICE definition.
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 or EXPIPE03	`/usr/lpp/cicsts/samples /webservices/wsbind /provider/inquireSingle.wsbind`
PLACE ORDER	EXORDRWS	EXPIPE01 or EXPIPE03	`/usr/lpp/cicsts/samples /webservices/wsbind /provider/placeOrder.wsbind`
DISPATCH STOCK	EXODRQWS	EXPIPE02 or EXPIPE04	`/usr/lpp/cicsts/samples /webservices/wsbind /requester/dispatchOrder.wsbind`

Function	WEBSERVICE name	PIPELINE attribute	WSBind attribute
DISPATCH STOCK endpoint (optional)	EXODEPWS	EXPIPE01 or EXPIPE03	/usr/lpp/cicsts/samples /webservices/wsbind /provider/dispatchOrderEndpoint.wsbind

Creating the URIMAP resources with resource definition online (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).

Before you begin

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 pickup directory of the PIPELINE. This ensures that the WEBSERVICE and URIMAP resources are not dynamically installed during a pipeline scan of the pickup directory. Alternatively, you can ensure that no value is specified for WSDIR in the PIPELINE. However, if you do not specify a value for WSDIR, no pipeline scans of the pickup directory occur. Therefore, all WEBSERVICE and URIMAP resources have to be created and installed using RDO.

Procedure

1. Use the CICS Explorer to create a URIMAP definition for the inquire catalog function of the example application.
   a. Create a URIMAP definition in the CICS Explorer by selecting **Definitions** > **URI Map Definition**.
   b. Right-click in the URI Map Definitions view and select **New** to create a new URIMAP definition.
   c. Type `INQCURI` in the **Name** text box, and type `*` in the **Host** text box. You must enter the **Path** attribute before you can create the URIMAP definition. In the **Path** text box type `/exampleApp/inquireCatalog`. **Usage** must be set to **Pipeline**; the PIPELINE resource is `EXPIPE01` or `EXPIPE03` for Java-based pipelines.
   d. Click **Finish** to finish the URIMAP definition.
   e. Double-click the new URIMAP resource to open the Editor. In the **Attributes** tab in the Editor, set the **Web Service** attribute to `EXINQCWS` and **TCP/IP Service** to `SOAPPORT`.
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

3. 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

Completing the installation

To complete the installation, install the RDO group that contains your resource definitions.

Procedure

Right-click the resource group in the Resource Group Definitions window. Select **Install**. Make sure that your CICSplex is correct and that you select your target region, then click **OK**.

Results

Your RDO is now installed and the application is ready for use.

Configuring the web client

Before you can use the web client, you must deploy the Java web archive (WAR) for the web client front end into one of the supported environments and configure it to call the appropriate endpoints in your CICS system.

About this task

The following environments are supported for the two versions of web client front end application ExampleAppClientV7.war and ExampleAppWrapperClientV7.war:

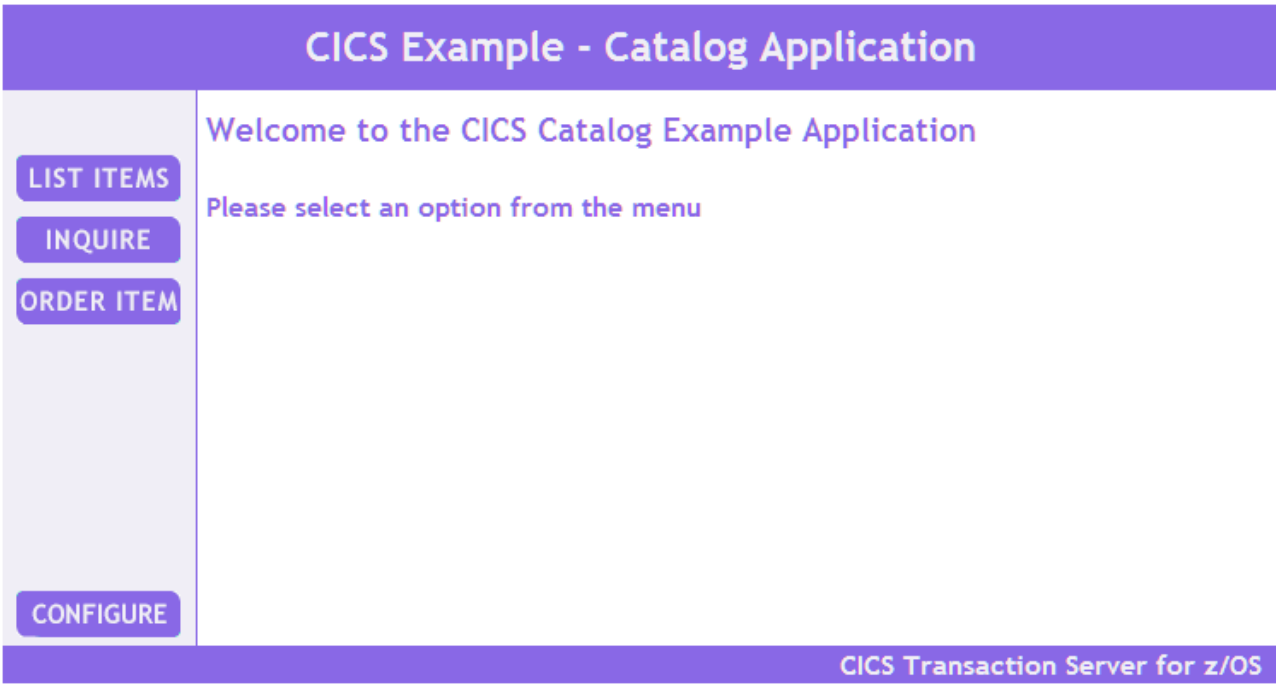
- CICS Liberty JVM server with WebSphere Liberty Profile V8.5.5

The WAR files are located in the *hlq*`/samples/webservices/client` directory in z/OS UNIX.

Procedure

1. To start the web client enter the following URL in your web browser, where *mylibertyserver* is the host name of the JVM Liberty Server on which the web client is installed.
   - For ExampleAppClientV7.war, use URL http://mylibertyserver:9080/ExampleAppClientV7/
   - For ExampleAppWrapperClientV7.war, use URL http://mylibertyserver:9080/ExampleAppWrapperClientV7/

   The example application displays the following page:



2. Click **CONFIGURE** to display the configuration page. The configuration page is displayed.
3. Enter the new endpoint for the web service of Inquire catalog, Inquire item, and 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 `8080` with the port number configured in the TCPIPSERVICE definition resource.
4. Click **SUBMIT**.

Results

The web application is now ready to run.

What to do next

The URL for the web services invocation is stored in an HTTP session. It is therefore necessary to repeat this configuration step each time a web browser is first connected to the client.

Running the web service enabled application

You can invoke the example application from a web browser.

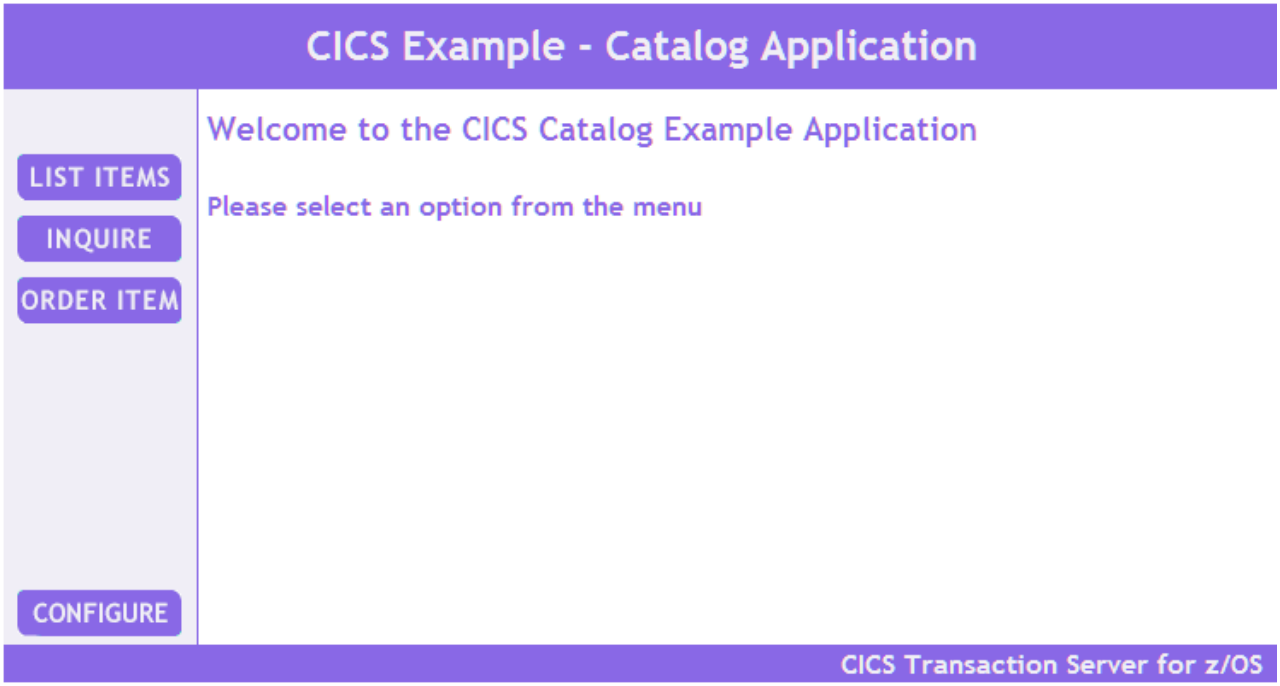
About this task

Please make sure that you configured the web client before proceed. See, Configuring the example application

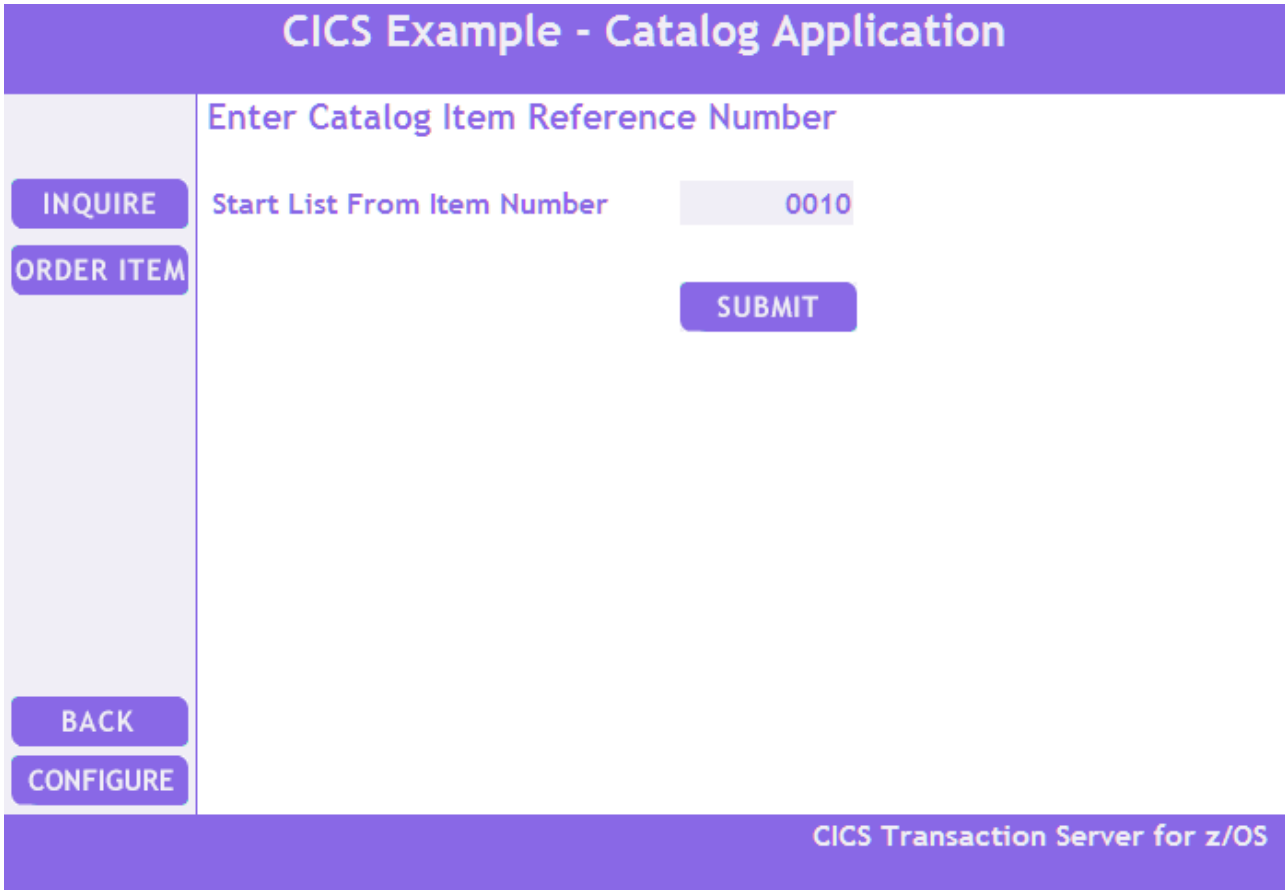
Procedure

1. Enter the following URL in your web browser: `http://myserver:9080/ExampleAppClientWeb/V7`, where *myserver* is the host name of the server on

which the web service client is installed. The example application displays the following page:

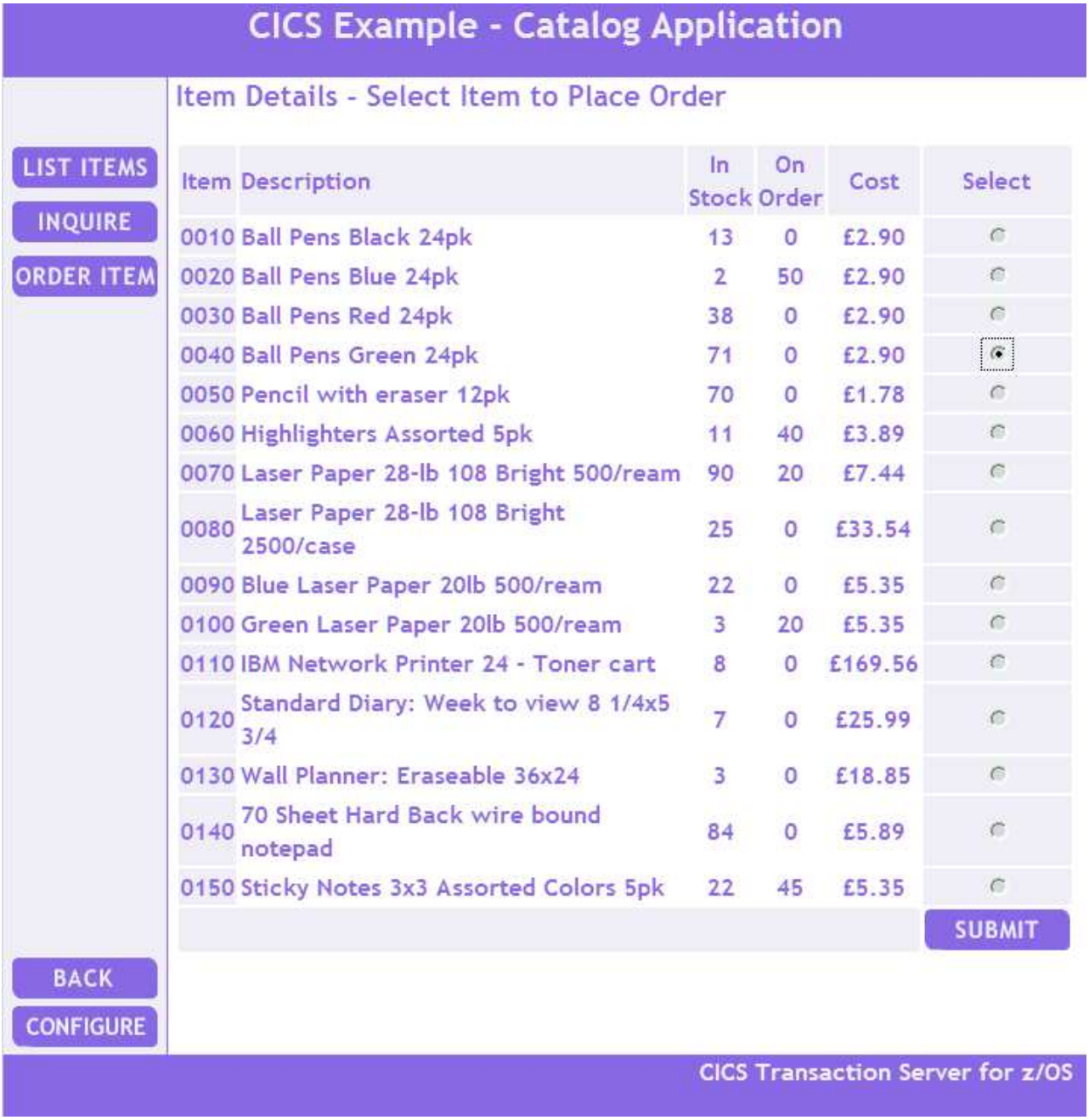


2. Click the **INQUIRE** button. The example application displays the following page:



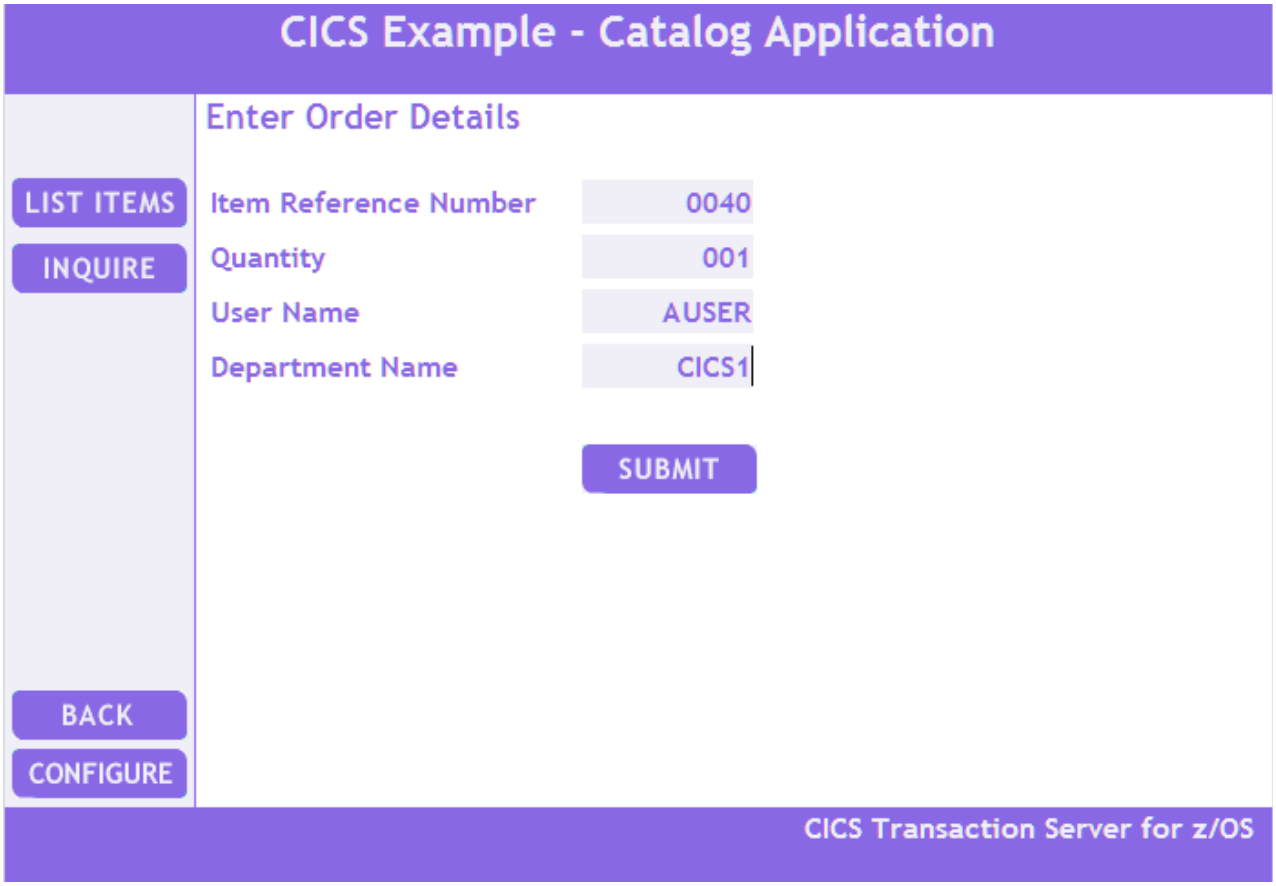
3. Enter an item number, and click the **SUBMIT** button.

   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.



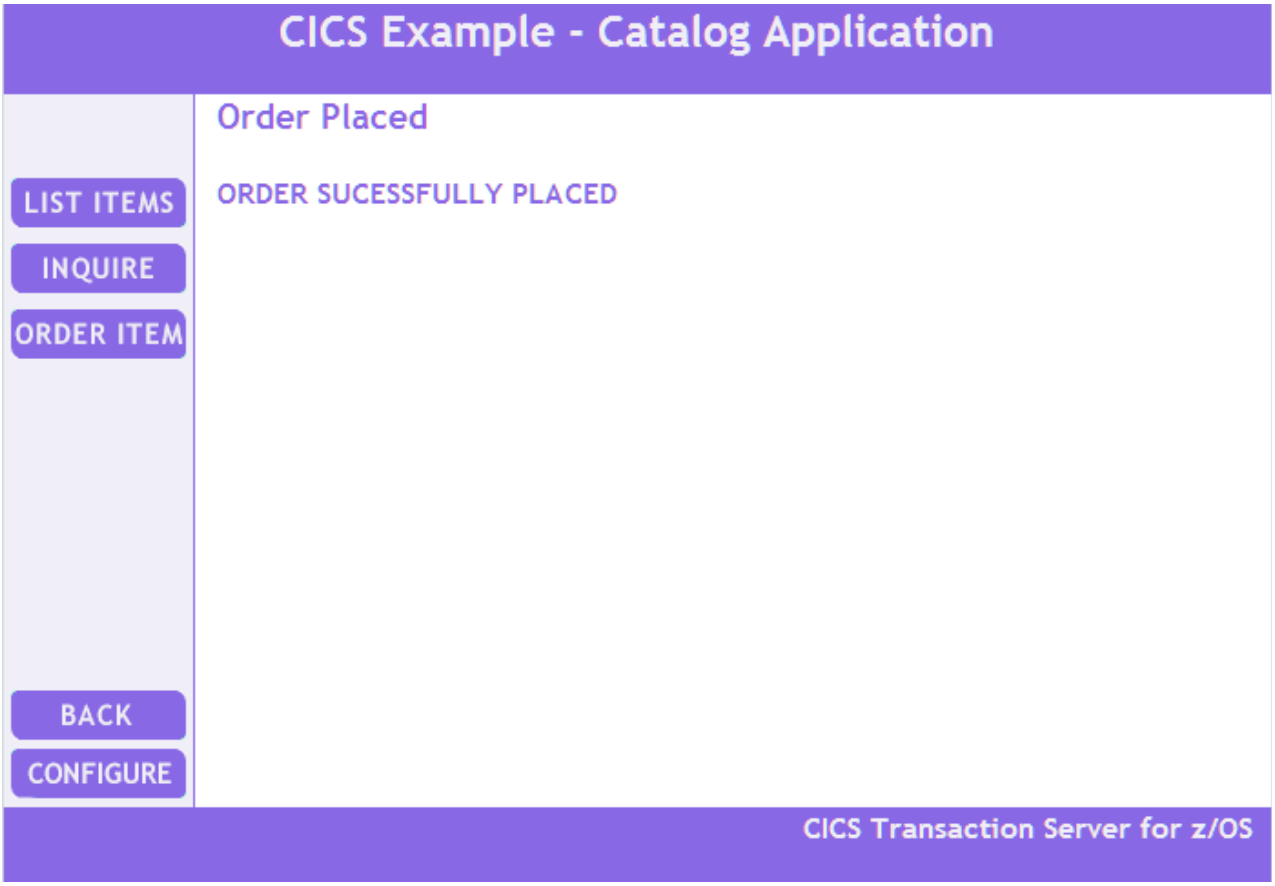
4. Select the item that you want to order.
   a. Click the radio button in the **Select** column for the item you want to order.
   b. Click the **SUBMIT** button.

   The application displays the following page:



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:



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 works without performing this task, you must perform a similar task if you want to deploy your own applications to extend the example application.

Extracting the program interface

To deploy a program with the CICS web services assistant, you must create a copybook that matches the COMMAREA or container interface.

About this task

In this example, the INQUIRE SINGLE ITEM function of the central catalog manager program (DFH0XCMN) is deployed as a web service. The interface to this program is a COMMAREA; the structure of the COMMAREA is defined in the copybook 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 three separate interfaces for the INQUIRE CATALOG, INQUIRE SINGLE ITEM, and 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. The copybook is now suitable for use with the web services assistant.

The 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 that 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.

Procedure

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 following parameters are required 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=mycicsserver:myport/exampleApp/inquireSingle
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
*/
```

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 software support organization.

PDSLIB=CICSHLQ.SDFHSAMP
The name of the partitioned data set (PDS) where the web services assistant looks for copybooks that define the request and response structures. In the example, this is the CICS installed data set SDFHSAMP.

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 is started when a web service request is received.

PGMINT=COMMAREA
The target program is invoked with a COMMAREA interface.

URI=*mycicsserver*:*myport*/exampleApp/inquireSingle

The unique part of the URI that is used in the generated web service definition, and used to create the URIMAP resource that maps incoming requests to the correct web service. The value specified results 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 resource has been installed.

Note: The parameter does not have a leading '/'.

WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
The location on z/OS UNIX to which the web service binding file is written.

Note: If the file is to be used with the pipeline scanning mechanism it must have the extension `.wsbind`.

WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
The location on z/OS UNIX to which the file containing the generated web service description is written. It is good practice to use matching names for the web service binding file and its corresponding web service description.

Conventionally, files containing web service descriptions have the extension `.wsdl`.

The web services description provides the information that a client must use 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 are created in the locations specified.

An example of the generated WSDL document

An example of the web service description (WSDL) document that is generated when the web services assistant program DFHLS2WS is run.

```
<?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: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 TCPIPSERVICE. -->
            <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 Websphere MQSeries.
                 Please replace "my-queue" with the appropriate queue name. -->
            <!-- <soap:address location="jms:/queue?destination=my-queue&amp;connectionFactory=()&amp;
            targetService=/exampleApp/inquireSingles.log&amp;initialContextFactory=com.ibm.mq.jms.Nojndi" /> -->
        </port>
    </service>
</definitions>
```

Deploying the web services binding file

The WEBSERVICE binding file, created by DFHLS2WS, is deployed into your CICS region dynamically when you install a PIPELINE resource.

About this task

When a pipeline scan command is issued, CICS scans the pickup directory to search for WEBSERVICE binding files 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 resource and the PIPELINE onto which the web service is installed. When a scanned WEBSERVICE resource is discarded, the URIMAP resource associated with it is also discarded.

Procedure

1. Modify the PIPELINE definition for your provider pipeline PIPELINE(EXPIPE01) in the CICS Explorer by selecting **Definitions** > **Pipeline Definitions**. Double-click EXPIPE01 to open the Pipeline Definition (EXPIPE01) editor. In the **Attributes** tab, change the **WS Directory** parameter to

/u/exampleapp/wsbind. This pickup directory contains the WEBSERVICE binding file that you generated with DFHLS2WS.

2. Copy any other WEBSERVICE binding files used by the application to the same directory. In this example, the following files are copied:

   ```
   inquireCatalog
   placeOrder
   ```

   They are provided in directory /usr/lpp/cicsts/samples/webservices/wsbind/provider.

3. Install the PIPELINE resource.

Results

CICS creates two URIMAP resources; the first URIMAP definition is required in a service provider when it contains information that maps the URI of an inbound web service request to the other resources (such as the PIPELINE resource) that service the request. The second URIMAP contains information that maps the URI of an inbound request for the WSDL document or documents associated with the web service.

Components of the base application

Use these tables to understand the components of the base application and the members supplied in the SDFHSAMP sample. The SDFHSAMP members listed contain BMS maps, COBOL source, and copybooks for the base application, web service client application, and the wrapper modules.

Table 23. 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 copybook generated by assembling DFH0XS1. DFH0XGUI and DFH0XCUI include this copybook
DFH0XM2U	COBOL copybook generated by assembling DFH0XS2 and editing the result to include an indexed array structure for ease of copybook programming. DFH0XGUI and DFH0XCUI include this copybook.
DFH0XM3	COBOL copybook generated by assembling DFH0XS3. DFH0XCFG includes this copybook

Table 24. 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 the controller program.

Table 24. SDFHSAMP members containing COBOL source for the base application (continued)

Member name	Description
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. This program 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. This program sets the text "`Order in dispatch`" in the return COMMAREA.
DFH0XSDS	A *stubbed* or dummy version of the data store program that allows the application to work when the VSAM catalog file has not been set up. DFH0XSDS 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. DFH0XSOD is used when outbound web services are not required.
DFH0XSSM	A stubbed version of the stock manager (replenishment) program. DFH0XSSM sets the return code in the COMMAREA to 0 and returns to its caller.
DFH0XVDS	The VSAM version of the data store program. DFH0XVDS accesses the VSAM file to perform reads and updates of the catalog.
DFH0XWOD	The web service version of the order dispatch program. DFH0XWOD issues an EXEC CICS INVOKE WEBSERVICE to make an outbound web service call to an order dispatcher.

Table 25. SDFHSAMP members containing COBOL copybooks for the base application

Member name	Description
DFH0XCP1	Defines a COMMAREA structure that 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 copybook.
DFH0XCP2	Defines a COMMAREA structure for the order dispatcher and stock manager modules. Programs DFH0XCMN, DFH0XSOD, DFH0XSSM, and DFH0XWOD include this copybook
DFH0XCP3	Defines a data structure for an inquire catalog request and response. Used as input to DFHLS2WS in order to produce `inquireCatalog.wsdl` and `inquireCatalog.wsbind`.
DFH0XCP4	Defines a data structure for an inquire single request and response. Used as input to DFHLS2WS in order to produce `inquireSingle.wsdl` and `inquireSingle.wsbind`.
DFH0XCP5	Defines a data structure for a place order request and response. Used as input to DFHLS2WS in order to produce `placeOrder.wsdl` and `placeOrder.wsbind`.
DFH0XCP6	Defines a data structure for a dispatch order request and response. Used as input to DFHLS2WS in order to produce `dispatchOrder.wsdl` and `dispatchOrder.wsbind`.
DFH0XCP7	Defines the data structure for a dispatch order request. Programs DFH0XODE and DFH0XWOD include this copybook
DFH0XCP8	Defines the data structure for a dispatch order response. Programs DFH0XODE and DFH0XWOD include this copybook.

Table 26. SDFHSAMP members containing COBOL source code for the web service client application thatruns 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 web service specified is one of the following:
	`inquireCatalogClient`
	`inquireSingleClient`
	`placeOrderClient`

Table 27. SDFHSAMP members containing COBOL copybooks for the web service client application thatruns 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 28. 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 29. SDFHSAMP members containing COBOL copybooks for the wrapper modules

Member name	Description
DFH0XWC1	Defines the data structure for the inquire catalog request. Program DFH0XICW includes this copybook.
DFH0XWC2	Defines the data structure for the inquire catalog response. Program DFH0XICW includes this copybook.
DFH0XWC3	Defines the data structure for the inquire single request. Program DFH0XISW includes this copybook.
DFH0XWC4	Defines the data structure for the inquire single response. Program DFH0XISW includes this copybook.
DFH0XWC5	Defines the data structure for the place order request. Program DFH0XPOW includes this copybook.
DFH0XWC6	Defines the data structure for the place order response. Program DFH0XPOW includes this copybook

Table 30. 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 type checking and error recovery that the catalog manager performs is limited.

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

Data is passed between the sample client and server programs by using a standard CICS communications area (COMMAREA).

The following code extract shows the catalog manager application COMMAREA 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).
           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.

Table 31. Catalog manager 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
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

Table 31. Catalog manager return codes (continued)

Type	Code	Explanation
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
A string that identifies the operation.

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
A string that identifies the operation.

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
A string that identifies the operation.

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 `01DSPO`.

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
A string that identifies the operation.

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 `01STKO`.

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
A string that identifies the operation.

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 that contains all information relating to the product inventory.

Catalog file records

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 that contains information used to configure the example application.

Configuration file records

The configuration file is a KSDS VSAM file with four distinct records.

Table 32. 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	

Table 32. General information record (continued)

Name	COBOL data type	Description
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	
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 33. 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 34. 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 35. 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.

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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 want 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.

Privacy Policy Considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

CICSPlex SM Web User Interface :

For the WUI main interface: Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the WUI Data Interface: Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the WUI Hello World page: Depending upon the configurations deployed, this Software Offering may use session cookies that collect no personally identifiable information. These cookies cannot be disabled.

For CICS Explorer: Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www-01.ibm.com/software/info/product-privacy/.

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 or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Bibliography

CICS books for CICS Transaction Server for z/OS

General

CICS Transaction Server for z/OS Program Directory - base, GI13-3375
CICS Transaction Server for z/OS Program Directory activation module - base, GI13-3376
CICS Transaction Server for z/OS Program Directory activation module - Developer Trial, GI13-3377
CICS Transaction Server for z/OS Program Directory activation module - Value Unit Edition, GI13-3378
CICS Transaction Server for z/OS What's New, GC34-7437
CICS Transaction Server for z/OS Upgrading to CICS TS Version 5.3, GC34-7436
CICS Transaction Server for z/OS Installation Guide, GC34-7414

Access to CICS

CICS Internet Guide, SC34-7416
CICS Web Services Guide, SC34-7452

Administration

CICS System Definition Guide, SC34-7428
CICS Customization Guide, SC34-7404
CICS Resource Definition Guide, SC34-7425
CICS Operations and Utilities Guide, SC34-7420
CICS RACF Security Guide, SC34-7423
CICS Supplied Transactions, SC34-7427

Programming

CICS Application Programming Guide, SC34-7401
CICS Application Programming Reference, SC34-7402
CICS System Programming Reference, SC34-7429
CICS Front End Programming Interface User's Guide, SC34-7412
CICS C++ OO Class Libraries, SC34-7405
CICS Distributed Transaction Programming Guide, SC34-7410
CICS Business Transaction Services, SC34-7403
Java Applications in CICS, SC34-7417

Diagnosis

CICS Problem Determination Guide, GC34-7422
CICS Performance Guide, SC34-7421
CICS Messages and Codes Vol 1, GC34-7418
CICS Messages and Codes Vol 2, GC34-7419
CICS Diagnosis Reference, GC34-7409
CICS Recovery and Restart Guide, SC34-7424
CICS Data Areas, GC34-7406
CICS Trace Entries, SC34-7430
CICS Debugging Tools Interfaces Reference,GC34-7408

Communication

CICS Intercommunication Guide, SC34-7415
CICS External Interfaces Guide, SC34-7411

Databases

CICS DB2® Guide, SC34-7407

CICS IMS™ Database Control Guide, SC34-7413

CICS Shared Data Tables Guide, SC34-7426

CICSPlex SM books for CICS Transaction Server for z/OS

General

CICSPlex SM Concepts and Planning, SC34-7441
CICSPlex SM Web User Interface Guide, SC34-7451

Administration and Management

CICSPlex SM Administration, SC34-7438
CICSPlex SM Operations Views Reference, SC34-7447
CICSPlex SM Monitor Views Reference, SC34-7446
CICSPlex SM Managing Workloads, SC34-7444
CICSPlex SM Managing Resource Usage, SC34-7443
CICSPlex SM Managing Business Applications, SC34-7442

Programming

CICSPlex SM Application Programming Guide, SC34-7439
CICSPlex SM Application Programming Reference, SC34-7440

Diagnosis

CICSPlex SM Resource Tables Reference Vol 1, SC34-7449
CICSPlex SM Resource Tables Reference Vol 2, SC34-7450
CICSPlex SM Messages and Codes, GC34-7445
CICSPlex SM Problem Determination, GC34-7448

Other CICS publications

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 5 Release 3 .

Designing and Programming CICS Applications, SR23-9692

CICS Application Migration Aid Guide, SC33-0768

CICS Family: API Structure, SC33-1007

CICS Family: Client/Server Programming, SC33-1435

CICS Family: Interproduct Communication, SC34-6853

CICS Family: Communicating from CICS on System/390, SC34-6854

CICS Transaction Gateway for z/OS Administration, SC34-5528

CICS Family: General Information, GC33-0155

CICS 4.1 Sample Applications Guide, SC33-1173

CICS/ESA 3.3 XRF Guide , SC33-0661

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

A

B

C

D

E

T

U

V

W

X

Z

Readers' Comments — We'd Like to Hear from You

CICS Transaction Server for z/OS
Version 5 Release 3
Web Services Guide

Publication No. SC34-7452-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your 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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: +44 1962 816151
- Send your comments via email to: idrcf@uk.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-7452-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM United Kingdom Limited
User Technologies Department (MP189)
Hursley Park
Winchester
Hampshire
United Kingdom
SO21 2JN

Fold and Tape | **Please do not staple** | Fold and Tape

SC34-7452-00

Cut or Fold
Along Line

SC34-7452-00