

CICS Transaction Server for z/OS  
Version 6

*Using Web Services with CICS*



**Note**

Before using this information and the product it supports, read the information in [Product Legal Notices](#).

This edition applies to the IBM® CICS® Transaction Server for z/OS®, Version 6 (product number 5655-BTA) and to all subsequent releases and modifications until otherwise indicated in new editions.

The beta version of IBM CICS Transaction Server for z/OS might be referred to in the product and documentation as CICS TS for z/OS, beta or 6.3.

© **Copyright International Business Machines Corporation 1974, 2026.**

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

---

# Contents

<b>About this PDF.....</b>	<b>vii</b>
<b>Chapter 1. CICS and web services.....</b>	<b>1</b>
CICS and SOAP web services .....	3
Planning to use SOAP web services.....	4
Message handlers and pipelines.....	5
SOAP nodes.....	11
SOAP messages and the application data structure.....	11
WSDL and the application data structure.....	14
WSDL and message exchange patterns.....	15
The web service binding file.....	16
SOAP architecture and message format.....	17
CICS and JSON web services.....	25
Concepts of JSON web services.....	26
Concepts of RESTful JSON web services.....	28
Planning a JSON service provider application.....	30
Planning a JSON service requester application.....	32
<b>Chapter 2. Configuring web services in CICS.....</b>	<b>33</b>
Configuring your CICS system for web services.....	33
CICS resources for web services.....	33
Web services discovery.....	36
Configuring CICS to use the IBM MQ transport for SOAP web services.....	36
Interoperability between the web services assistant and WSRR.....	41
Creating the web services infrastructure.....	42
The web services infrastructure.....	43
Creating the CICS infrastructure for a SOAP service provider.....	53
Creating the CICS infrastructure for a SOAP service requester.....	55
Creating the CICS infrastructure for a JSON service provider.....	57
Creating the CICS infrastructure for a non-Java JSON service provider.....	58
Message handlers and the message handler interface.....	59
SOAP message handlers.....	66
Application handlers.....	70
Supplying your own message handlers.....	72
Pipeline configuration files.....	72
Support for Web Services transactions.....	81
Support for MTOM/XOP optimization of binary data.....	89
Support for Web Services Addressing.....	97
Support for WS-Security.....	117
Controlling runtime processing for web services.....	117
<b>Chapter 3. Developing web services.....</b>	<b>127</b>
Developing JSON web services.....	127
Creating a JSON service provider application.....	128
Creating a JSON web service client application.....	137
JSON web service restrictions.....	138
Developing SOAP web services.....	139
The CICS web services assistant.....	140
Creating a web service provider using the assistant.....	171
Creating a web service requester using the assistant.....	181

Developing web services using tooling.....	183
Creating your own XML-aware web service applications.....	183
Testing CICS SOAP Web services.....	186
Validating SOAP messages.....	191
Using Java with web services .....	192
Deploying a Java provider-mode web service in an Axis2 JVM server.....	192
Creating a web service that generates and parses XML.....	194
Creating a web service that has a COBOL interface.....	194
Deploying a requester-mode JAX-WS web service.....	194
Deploying a Java provider-mode web service in a Liberty JVM server .....	195
<b>Chapter 4. Developing with JSON.....</b>	<b>197</b>
The CICS JSON assistant .....	197
DFHLS2JS: High-level language to JSON schema conversion for request-response services.....	198
DFHJS2LS: JSON schema to high-level language conversion for request-response services.....	210
DFHJS2LS: JSON schema to high-level language conversion for RESTful services.....	222
DFHLS2JS: High-level language to JSON schema conversion for linkable interface.....	234
DFHJS2LS: JSON schema to high-level language conversion for linkable interface.....	243
How the CICS JSON assistant maps between high-level languages and JSON schemas.....	253
Generating mappings from language structure.....	301
Generating mappings from a JSON schema.....	302
Transforming application data to JSON by linking to DFHJSON.....	304
Transforming application data to JSON by using the <b>TRANSFORM DATATOJSON</b> API command.....	305
Transforming JSON to application data by linking to DFHJSON.....	306
Transforming JSON to application data by using the <b>TRANSFORM JSONTODATA</b> API command.....	307
Handling invalid and uninitialized application-supplied data.....	307
Example 1: toleration of decimal fields.....	308
<b>Chapter 5. Developing with XML.....</b>	<b>311</b>
The CICS XML assistant.....	311
DFHLS2SC: high-level language to XML schema conversion.....	312
DFHSC2LS: XML schema to high-level language conversion.....	320
How the CICS assistants map between high-level languages and XML schemas.....	329
Querying XML from an application.....	383
Handling XML by data type.....	384
Handling <xsd:any> data types.....	385
Handling variably repeating content in COBOL.....	385
Generating mappings from language structures.....	388
Generating mappings from an XML schema.....	390
Transforming application data to XML.....	391
Transforming XML to application data.....	392
Validating XML transformations.....	393
<b>Chapter 6. Security for SOAP web services.....</b>	<b>395</b>
How it works: SOAP message security .....	395
How it works: Authentication for CICS with SOAP message security.....	399
How it works: Signing SOAP messages .....	401
How it works: SOAP message encryption .....	402
Designing security for CICS web service providers.....	405
Design example: Securing a direct request with TLS client authentication.....	408
Design example: Asserting an identity to the CICS web service provider.....	413
Designing security for CICS web service requesters.....	417
Design example: Securing a web service request with TLS client authentication.....	419
Design example: Securing a web service request with identity assertion.....	422
Configuring SOAP message security for CICS web services.....	426
Installing the prerequisites for WS-Security support.....	427
Configuring the pipeline for WS-Security.....	428

Configuring provider mode web services for identity propagation.....	431
Configuring RACF for WS-Security.....	433
Invoking the Trust client from a message handler.....	435
Writing a custom security handler.....	436

**Chapter 7. Troubleshooting web services..... 439**

Troubleshooting SOAP web services.....	439
Diagnosing deployment errors.....	439
Diagnosing service provider runtime errors.....	441
Diagnosing service requester runtime errors.....	442
Diagnosing MTOM/XOP errors.....	444
Diagnosing data conversion errors.....	445
Troubleshooting JSON web services.....	447
JSON deployment problems.....	448
JSON assistant problems.....	449
Troubleshooting problems with JSON requests.....	449
JSON error responses.....	450
JSON assistant return codes.....	451

**Appendix A. Pipeline configuration elements.....453**

<addressing>.....	454
<apphandler_class>.....	455
<authentication> .....	456
<auth_token_type>.....	460
<cics_json_handler_java>.....	461
<cics_mtom_handler>.....	462
<cics_soap_1.1_handler>.....	463
<cics_soap_1.1_handler_java>.....	465
<cics_soap_1.2_handler>.....	467
<cics_soap_1.2_handler_java>.....	469
<default_http_transport_handler_list>.....	471
<default_mq_transport_handler_list>.....	472
<default_transport_handler_list>.....	472
<dfhwsse_configuration> .....	473
<dfhmtom_configuration>.....	475
<encrypt_body>.....	475
<handler>.....	476
<jvmserver>.....	477
<mime_options>.....	477
<mtom>.....	478
<mtom_options>.....	478
<named_transport_entry>.....	479
<provider_pipeline>.....	480
<provider_pipeline_json>.....	481
<repository>.....	482
<requester_pipeline>.....	482
<service>.....	483
<service_handler_list>.....	483
<service_parameter_list>.....	484
<sign_body>.....	485
<sts_authentication>.....	485
<sts_endpoint>.....	488
<terminal_handler>.....	489
<transport>.....	490
<transport_handler_list>.....	491
<wsse_handler>.....	491
<xop_options>.....	492

<b>Appendix B. Containers used in web services pipelines.....</b>	<b>495</b>
Control containers.....	495
DFHFUNCTION container .....	499
Context containers.....	501
Using the DFHWS-URI container.....	509
Header processing program containers.....	512
Security containers.....	513
SAML support containers.....	515
Containers generated by CICS.....	518
<b>Appendix C. JSON transformer linkable interface containers.....</b>	<b>519</b>
<b>Appendix D. Web services samples.....</b>	<b>523</b>
The CICS catalog manager example application.....	523
The base application.....	523
Installing and setting up the base application.....	526
Running the example application with the BMS interface.....	531
Web service support for the example application .....	533
Configuring the web client.....	542
Running the web service enabled application.....	543
Deploying the example application.....	543
Components of the base application.....	548
File structures and definitions.....	555
JSON samples.....	558
Example HTTP GET request using a query string.....	558
Example HTTP request with a JSON body.....	558
<b>Notices.....</b>	<b>559</b>
<b>Index.....</b>	<b>565</b>

## About this PDF

---

This PDF describes how to use web services in CICS. It is aimed at system programmers who are responsible for configuring CICS to support web services, and application developers who are responsible for applications that will be deployed in a web services environment. Before CICS TS V5.4, this PDF was called the "Web Services Guide".

For details of the terms and notation used in this book, see [Conventions and terminology used in the CICS documentation](#) in IBM Documentation.

### **Date of this PDF**

This PDF was created on 2026-04-07 (Year-Month-Date).



---

# Chapter 1. CICS and web services

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.

CICS support for web services conforms to a number of industry standards and specifications. The supported industry standards and specifications are listed in [Supported standards](#).

## CICS and z/OS Connect

z/OS Connect is a separately-orderable IBM product that runs in a standalone environment.

z/OS Connect provides RESTful API access to z/OS applications and data hosted in subsystems such as CICS, IMS, IBM MQ and Db2®. The framework provides concurrent access, through a common interface, to multiple z/OS subsystems.

z/OS Connect also provides the capability that allows CICS, IMS and z/OS applications to access any RESTful endpoint, inside or outside z/OS, through RESTful APIs with JSON formatted messages.

To learn more, see [Running a z/OS Connect Server within CICS](#).

## Web services terminology

### Extensible Markup Language (XML)

A standard for document markup, which uses a generic syntax to mark up data with simple, human-readable tags. The standard is endorsed by the [World Wide Web Consortium \(W3C\)](#).

### 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 (WS-Security)**

A set of enhancements to SOAP messaging that provides message integrity and confidentiality. The specification is published by [OASIS](#). See [How it works: SOAP message security](#).

**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 \(W3C\)](#).

## CICS and SOAP web services

---

You can expose existing CICS applications as SOAP web services and create CICS applications to act as SOAP web service providers or web service requesters. 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, by using code that you write to suit your particular needs. Both approaches are underpinned by an infrastructure 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 that is used by your application.

For either approach, you can either use the supplied CICS message handlers or use your own message handlers to perform extra processing on your request and response messages.

## 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.

### **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 IBM Developer for z/OS 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.

For security considerations, see [Security for SOAP web services](#).

## 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.

### Phases in the operation of a pipeline

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

#### Request phase

During the *request phase*, CICS invokes each handler in the pipeline in turn. Each message handler can process the request before returning control to CICS.

#### Response phase

Following the request phase is 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 1 on page 6](#) shows a pipeline of three message handlers:

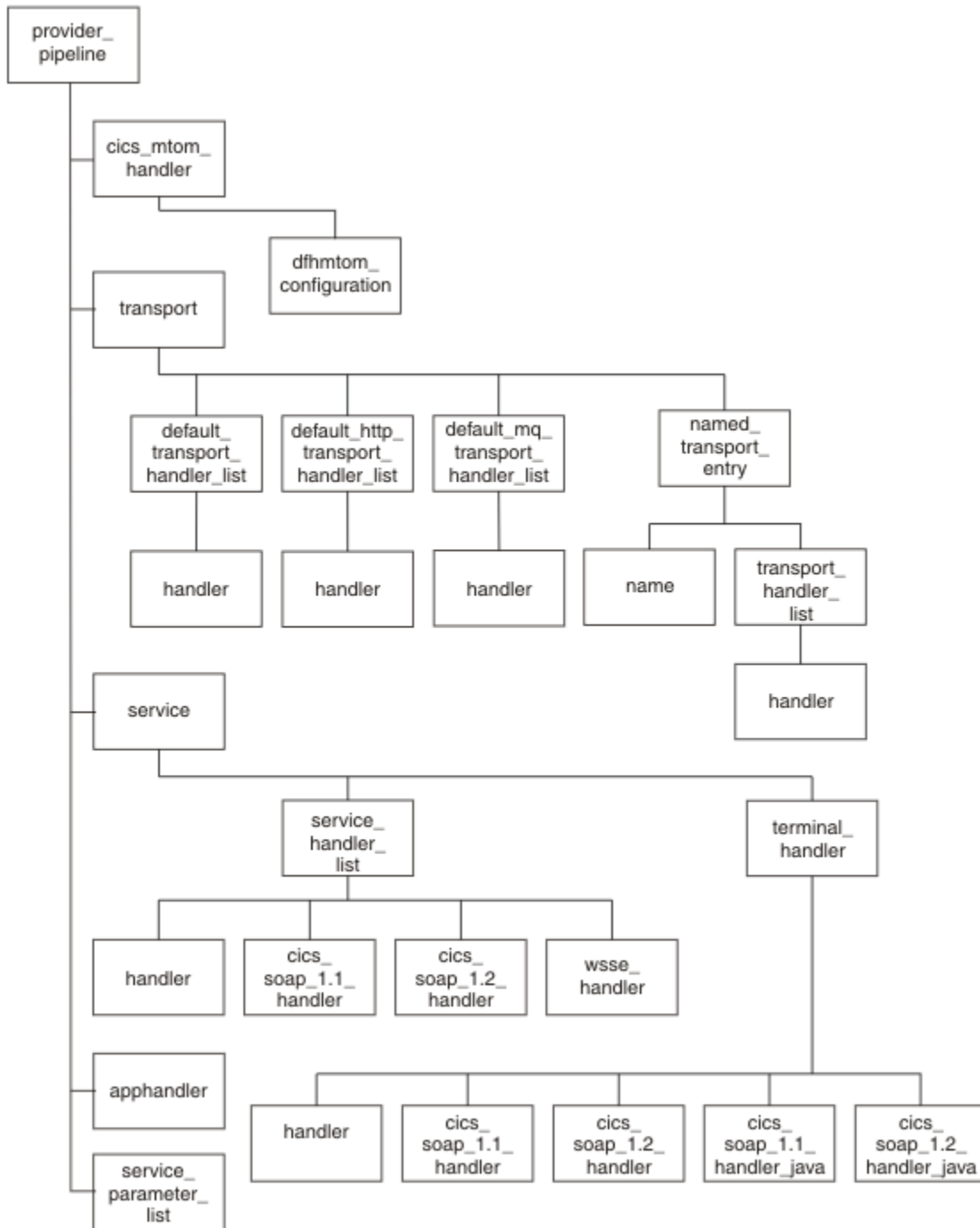


Figure 1. Example: A generic CICS pipeline

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

## Transition between the phases

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.

## 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.

Figure 2 on page 7 shows an example pipeline that contains three handlers, handler 1, handler 2 and handler 3. Suppose that handler 2 is responsible for performing security checks.

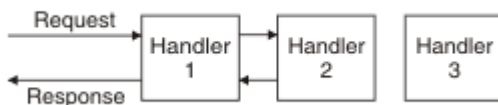


Figure 2. Example: Interrupting the pipeline flow

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.

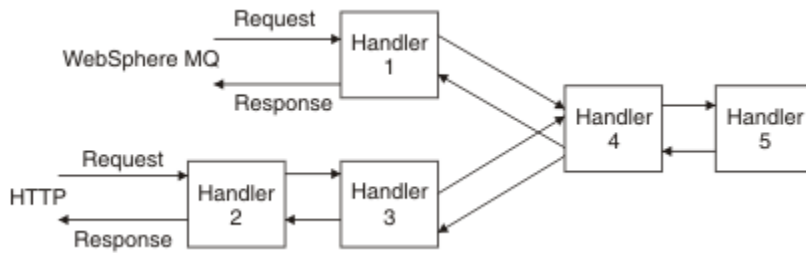
A handler that interrupts the normal message flow in this way must only do so if the originator of the message expects a response; for example, a handler should not generate a response when an application uses a one-way message flow from service requester to provider.

## 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 TCPIP SERVICE for the HTTP transport, a QUEUE for the MQ transport) is in use. This is illustrated in [Figure 3 on page 8](#).



*Figure 3. Example: A 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.

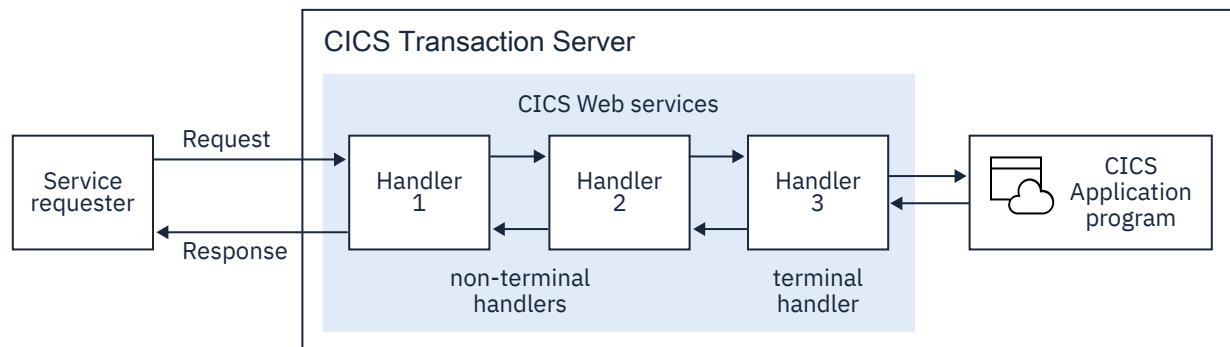
## A service provider pipeline

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 4 on page 8](#) illustrates a pipeline of three message handlers in a service provider setting:



*Figure 4. 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 5 on page 9 illustrates a pipeline of three message handlers in a service requester setting:

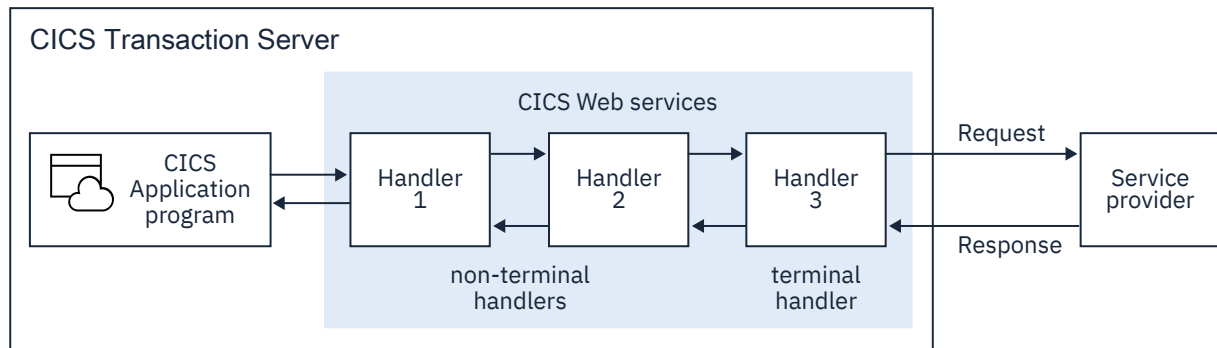


Figure 5. A service requester pipeline

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.

## 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 6 on page 11 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.

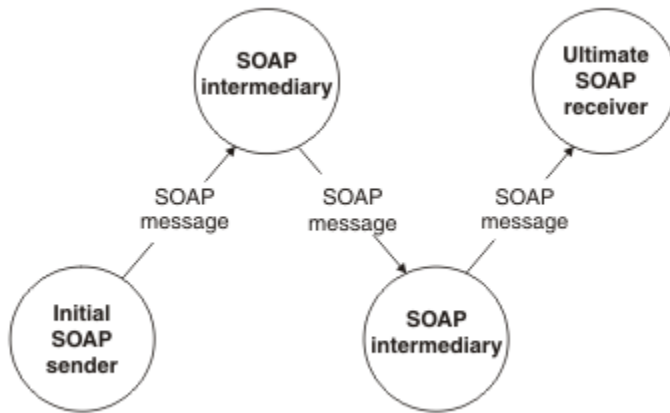


Figure 6. An example of a SOAP message path

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.

## 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.

## 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.

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 7 on page 12 shows where these files are used in a service provider.

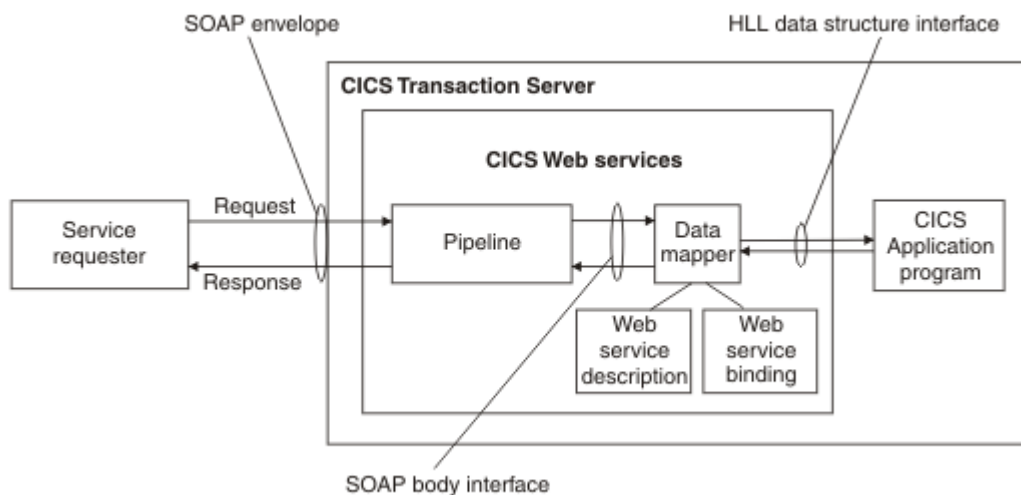


Figure 7. Mapping the SOAP body to the application data structure in a service provider

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 8 on page 13 shows where these files are used in a service requester.

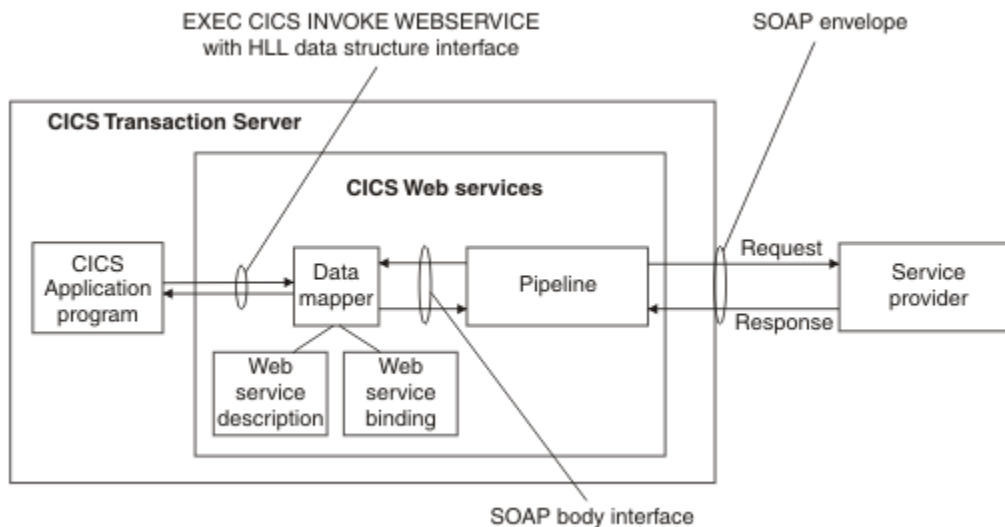


Figure 8. Mapping the SOAP body to the application data structure in a service requester

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, 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 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 9 on page 14 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 9. 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 10 on page 15](#):

```
<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 10. WSDL fragment derived from a COBOL data structure

## WSDL and message exchange patterns

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

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*: .

## **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.

## SOAP architecture and message format

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.

## 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.

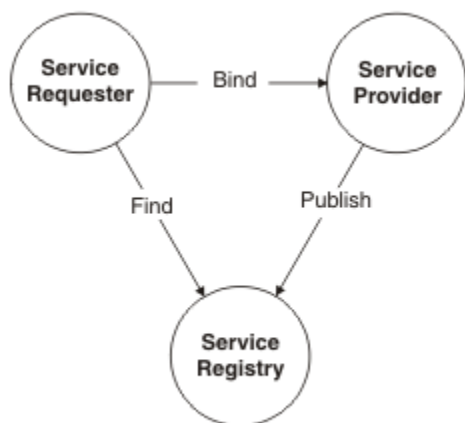


Figure 11. web services components and interactions

## 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

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 12 on page 19](#).

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.

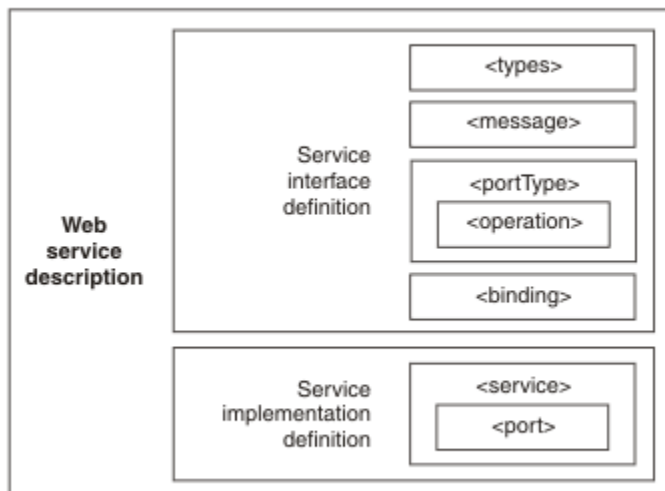


Figure 12. Structure of a web service description

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.

### **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. For details, see ["The SOAP header" on page 22.](#)

### The SOAP body

The SOAP <Body> is a mandatory subelement of the SOAP envelope, in which the main end-to-end information conveyed in a SOAP message is carried. It contains information intended for the ultimate recipient of the message.

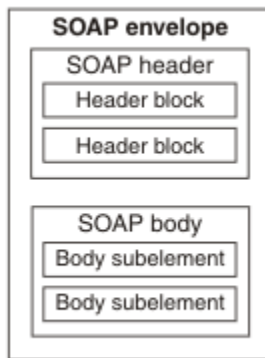
The <Body> element and its associated child elements are used to exchange information between the initial SOAP sender and the ultimate SOAP receiver. SOAP defines one child element for the <Body>: the <Fault> element, which is used for reporting errors. Other elements in the <Body> are defined by the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

### The SOAP fault

The SOAP <Fault> is a subelement of the SOAP body, which is used for reporting errors. For details, see [“The SOAP fault”](#) on page 23.

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 13](#) on page 21 shows the main elements of a SOAP message.



*Figure 13. The structure of a SOAP message*

[Figure 14](#) on page 22 is an example of a SOAP message that contains header blocks (the <m:reservation> and <n:passenger> elements) and a body (containing the <p:itinerary> and <q:lodging> elements).

```

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Áke 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 14. 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 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.

## CICS and JSON web services

There are several ways to get started with JSON 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 web services. CICS supports several different technologies for exposing resources as JSON based services. This section relates to an older technology called *JSON web services*.

**Note:** IBM's premiere technology for RESTful APIs that use JSON message formats is z/OS Connect. It is broadly compatible with CICS JSON web services, but offers many additional integration options and capabilities. To learn more about z/OS Connect, see [z/OS Connect Enterprise Edition product documentation](#).

JSON web services is a technology for enabling CICS programs as JSON services. These may be RESTful services, or request-response remote-procedure-call style services. The technology is derived from that used for SOAP web services and uses JCL procedures called DFHLS2JS and DFHJS2LS that are used to generate WSBind files. These WSBind files are deployed into CICS as WEBSERVICE resources. The WEBSERVICE resources facilitate automatic conversions between JSON and application binary data formats.

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

- Get familiar with [“Concepts of JSON web services” on page 26](#) and [“Concepts of RESTful JSON web services” on page 28](#).
- Plan for the deployment of an application as a service provider.

You might already know enough about how you will use web services in CICS to start planning your applications and the related infrastructure.

- If you want a hands-on way to learn about web services in CICS, CICS provides an example of a catalog management application, which you can enable as a JSON web service provider. To do this, use DFHLS2JS to generate a web service from the language structures provided. You could use a web browser or third party client application to test the JSON web service. For more information, see [Creating a service provider application from a data structure](#).

The example application is described in [The CICS catalog manager example application](#).

## Preliminary questions for planning

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

### 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 the CICS assistant to generate the mappings between your data structures and JSON schemas?

The assistant provides a rapid deployment of many applications into a JSON 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 JSON assistant. For example, if you have existing code that maps data structures to JSON messages, there is no advantage in reengineering your application with the JSON assistant.

Although the CICS 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 JSON schema mapping](#).

## Concepts of JSON web services

Read this topic to understand the concepts behind JSON web services.

### Web services

A *web service* is a generic term for a software function that is hosted at a network addressable location. In this general sense it can imply a Cloud based service, a Utility service or even a departmental application. The term *web service* can also be used in a more specific sense, such as a hosted service using SOAP which is described using a WSDL document. It is this more specific meaning that is usually implied by the term *web services in CICS*. However, the more general term is often used by the JSON community when describing JSON based services. JSON web services uses the term in its generic sense.

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 will be smaller than the equivalent XML message.
- JSON is easy to integrate in JavaScript applications, but XML isn't. This makes JSON a preferred data format with 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 results in fewer options for securing and configuring a workload.
- SOAP web services are described using WSDL documents. JSON web services are structured less formally; they tend to be loosely coupled and prefer documentation by example.
- SOAP web services have an explicit error format involving SOAP Fault messages. There's no equivalent for JSON.

There are also many similarities between JSON and SOAP:

- The CICS implementation of JSON is derived from the SOAP architecture, and shares many of the concepts and artifacts.
- Both involve offline utility programs that assist with mapping application data to and from the external data representation. For SOAP there is DFHLS2WS and DFHWS2LS, for JSON there is DFHLS2JS and DFHJS2LS.
- The deployment mechanism for both technologies involve a PIPELINE resource, a WEBSERVICE resource, and a URIMAP resource.

## JSON schema

One disadvantage of JSON compared to SOAP is the difficulty in documenting the structure of a JSON interface. SOAP web services have the advantage of WSDL documents, together with XML schemas. A WSDL document may not be easy to understand, but there are many tools available to work with WSDL documents.

The nearest equivalent for JSON is the JSON Schema specification available at <http://json-schema.org/>. At the time of writing this is a draft specification which is making its way through the IETF standardization process. The CICS JSON assistant (DFHLS2JS and DFHJS2LS) supplies a partial implementation of draft 4 of this emerging specification. DFHLS2JS can be used to generate JSON schema, and DFHJS2LS can be used to process them.

You can use the JSON schema to understand the valid syntax and content model for a JSON web service that has been implemented in CICS. The JSON schema specification doesn't have the same tooling ecosystem as the XML schema specification, but a new generation of JSON tools might emerge to use this data format.

## CICS implementation of JSON based web services

CICS supports three modes of JSON web service, z/OS Connect, Request-Response and RESTful. CICS also supports a programmatic scenario in which applications can transform JSON data to and from COBOL style data formats themselves.

### z/OS Connect

z/OS Connect is a separately-orderable IBM product that runs in a standalone environment.

z/OS Connect provides RESTful API access to z/OS applications and data hosted in subsystems such as CICS, IMS, IBM MQ and Db2. The framework provides concurrent access, through a common interface, to multiple z/OS subsystems.

z/OS Connect also provides the capability that allows CICS, IMS and z/OS applications to access any RESTful endpoint, inside or outside z/OS, through RESTful APIs with JSON formatted messages.

To learn more, see [Running a z/OS Connect Server within CICS](#).

### **Request-Response**

The Request-Response JSON pattern is very similar to that of SOAP based web services in CICS. The web service is implemented using a PROGRAM in CICS. The PROGRAM has input and output data formats, described using language structures (such as COBOL copybooks), and CICS is responsible for transforming incoming JSON messages into application data, and linking to the application. The application returns output data back to CICS, and CICS transforms this into JSON data to return to the client.

In this scenario the JSON client must connect to CICS using the HTTP POST method.

A Request-Response mode JSON web service can be developed in either bottom-up mode or top-down mode. In bottom-up mode an existing CICS PROGRAM is exposed as a JSON web service using the DFHLS2JS JSON Assistant. In top-down mode a new JSON web service can be developed to implement an interface described using existing JSON schemas. In top-down mode, the DFHJS2LS JSON Assistant is used to generate new language structures, and an application must be created to use them.

The Request-Response pattern may be used to build JSON Web Services that target either COMMAREA or channel-attached CICS PROGRAMs. A Request-Response JSON web service can be used only in provider mode (where CICS acts as the server).

### **RESTful**

This scenario is different to that of SOAP web services. The concept of a RESTful JSON web service is described more completely in [Concepts of RESTful JSON web services](#). A RESTful JSON web service implements the architectural principles of the REpresentational State Transfer (REST) design pattern. This design pattern is unlikely to be relevant for existing CICS applications, so is available only in top-down mode.

A JSON schema can be processed by DFHJS2LS in RESTful mode. An application must be written to implement the service and it will have to behave differently depending on the HTTP method that was used for the incoming request.

CICS implements a pure style of RESTful application, where the data format for POST (create) GET (inquire) and PUT (replace) are the same.

RESTful JSON web service applications must use a channel based program interface; COMMAREAs are not supported. A RESTful JSON web service can be used only in provider mode (where CICS acts as the server).

### **Programmatic mode**

In this scenario an application can use the EXEC CICS TRANSFORM command or LINK to a CICS supplied program, DFHJSON, and ask it to transform application data into JSON data, or JSON data into application data. For example, an application might use this facility to generate JSON data to send to a remote JSON web service. To do this, it must contact the remote JSON web service using the CICS WEB API. For more details, see [Developing with JSON](#).

CICS has no built-in support for requester mode JSON web services, but an application can call a remote JSON web service by exploiting the programmatic mode.

## **Concepts of RESTful JSON web services**

Read this topic to understand the concepts behind RESTful web services.

### **RESTful web services**

REpresentational State Transfer, or REST, is a design pattern for interacting with resources stored in a server. Each resource has an identity, a data type, and supports a set of actions.

The RESTful design pattern is normally used in combination with HTTP, the language of the internet. In this context the resource's identity is its URI, the data type is its Media Type, and the actions are made up of the standard HTTP methods (GET, PUT, POST, and DELETE).

This style of service differs from Request-Response style web services:

- Request-Response services start interaction with an Application, whereas RESTful services typically interact with data (referred to as 'resources').
- Request-Response services involve application defined 'operations', but RESTful services avoid application specific concepts.
- Request-Response services have different data formats for each message, but RESTful service typically share a data format across different HTTP methods.

The four major HTTP methods define the four operations that are commonly implemented by RESTful Services. The HTTP POST method is used for creating a resource, GET is used to query it, PUT is used to change it, and DELETE is used to destroy it. The most common RESTful architecture involves a shared data model that is used across these four operations. This data model defines the input to the POST method (create), the output for the GET method (inquire) and the input to the PUT method (replace). This simple design pattern is popular within the RESTful community, but it's not the only RESTful design pattern. The HTTP status code is used to indicate success or failure of the operation. Some RESTful APIs are designed in other ways.

A fifth HTTP method called 'HEAD' is sometimes supported by RESTful web services. This method is equivalent to GET, except that it returns only HTTP Headers, and no Body data. It's sometimes used to test the Existence of a resource. Not all RESTful APIs support use of the HEAD method.

Traditional CICS applications are unlikely to match the RESTful architectural pattern. Typical CICS applications implement multiple operations, each of which will have data models for input and output formats. These existing operations are unlikely to map directly to the four HTTP methods. For this reason the RESTful architectural pattern is primarily aimed at new applications in CICS. To expose existing CICS applications as RESTful Services you may need to wrap them with a new interface that conforms to the RESTful principles.

## The URI

The identity of a RESTful service is indicated by its URI. A URI can be made up of several components, including the host name, port number, the path, and an optional query string. The domain name and port number together target a TCPIPService resource in CICS. For more information, see [TCPIPService resources](#). The URI path is a qualifier, and might be sufficient to uniquely identify the service. However, many RESTful web services use an additional query string to identify the precise resource. Consider the following examples:

- `http://www.example.org:10000/JSONServices/AccountService`
- `https://www.example.org:10000/JSONServices?Service=Account`

In the first example the URI path is `JSONServices/AccountService`. In the second example the path is `JSONServices` and there is an additional query string of `Service=Account`. Both styles of URI are considered to be acceptable for JSON. This is an important difference compared to SOAP. Under SOAP the first style of URI is preferred.

CICS JSON services can be converted into RESTful services using z/OS Connect. A Graphical User Interface is used to map URI fragments and HTTP headers into fields of an existing copybook, and different programs can potentially act as the target for each HTTP method. The ability to construct RESTful services from existing application assets is one of the main advantages of z/OS Connect over the other JSON technologies in CICS.

CICS also has an older technology for implementing a limited form of RESTful service. A URIMAP resource can be used to identify the appropriate WEBSERVICE and PIPELINE to use when processing an inbound message. The URIMAP can map a URI to a specific PIPELINE or WEBSERVICE, potentially including the query string of the URI in that mapping

CICS uses a URIMAP resource to identify the appropriate WEBSERVICE and PIPELINE to use when processing an inbound message. The URIMAP supports use of a query string as part of the path attribute. Therefore the URIMAP is suitable for use with both types of URI.

## Planning a JSON 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 on page 30 shows a typical application which is partitioned to ensure a separation between communication logic and business logic.

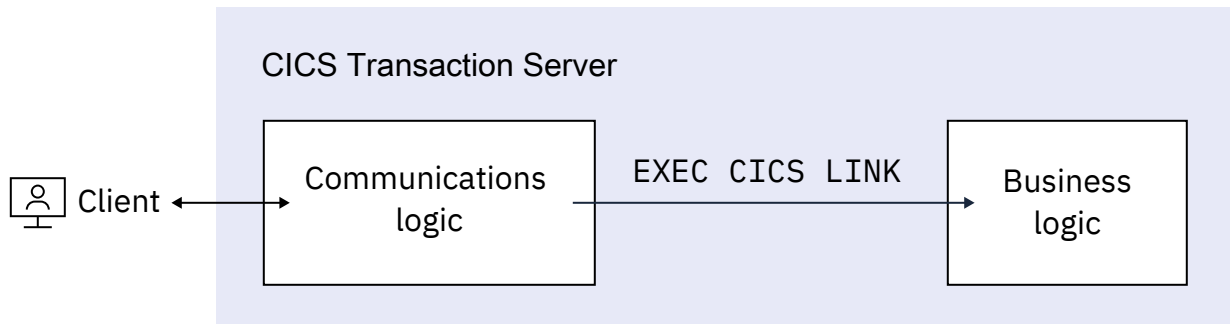


Figure 15. Application partitioned into communications and business logic

In many cases, you can deploy the business logic directly as a service provider application. This is illustrated in Figure 16 on page 30.

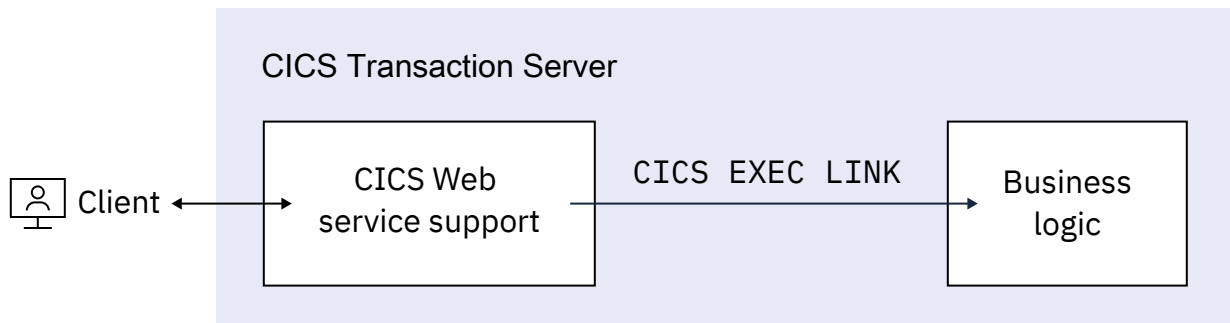


Figure 16. Simple deployment of CICS application as a web service provider

To use this simple model, the following conditions apply:

### **When using z/OS Connect to generate the mapping between a JSON schema or Swagger document and application data structures:**

z/OS Connect supports a broader range of application copybook structures than are supported by the CICS assistant. In most cases those copybooks will be supported unchanged. If not, you might interpose a wrapper program between the z/OS Connect service and your business logic. For more information, see [z/OS Connect Enterprise Edition product documentation](#).

### **When you are using the CICS assistant to generate the mapping between JSON schema and application data structures:**

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

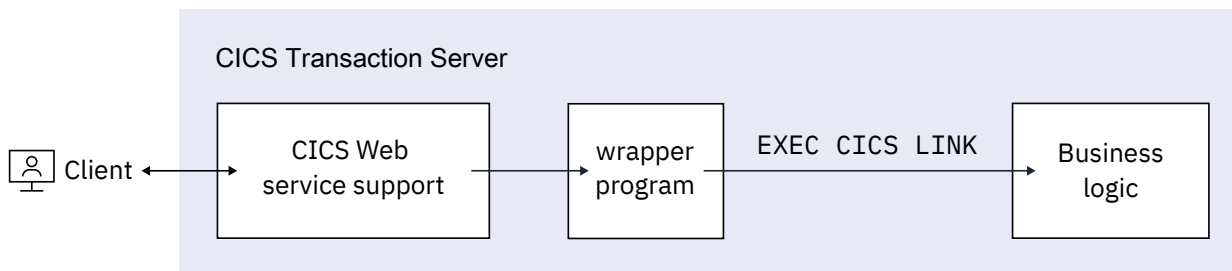
You will also need a wrapper program when you deploy an existing program to provide a service that conforms to an existing web service description: if you process the web service description using the assistant, the resulting data structures are very unlikely to match the interface to your business logic.

#### **When you are not using the CICS 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 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 assistant cannot map directly into a JSON 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 assistant, you should also consider how to handle rolling back changes when errors occur. When a JSON request message is received from a service requester, the JSON 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 JSON 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 JSON 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 JSON 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.

#### **Related information**

[“Planning a JSON service requester application” on page 32](#)

CICS does not provide built-in support for requester mode JSON web services. Those services are provided by z/OS Connect. To invoke a remote JSON web service from a CICS application without using z/OS Connect, use **EXEC CICS WEB** API commands.

## Planning a JSON service requester application

CICS does not provide built-in support for requester mode JSON web services. Those services are provided by z/OS Connect. To invoke a remote JSON web service from a CICS application without using z/OS Connect, use **EXEC CICS WEB** API commands.

You can write an application program to call a RESTful web service by using either the **TRANSFORM DATATOJSON** and **TRANSFORM JSONTODATA** API commands or the linkable interface, DFHJSON, to transform JSON and then using the WEB commands to send it to the remote service provider. For details, see [Creating a JSON web service client application](#).

For more information about z/OS Connect, see [z/OS Connect Enterprise Edition product documentation](#).

### Related concepts

[“Concepts of JSON web services” on page 26](#)

Read this topic to understand the concepts behind JSON web services.

[“Concepts of RESTful JSON web services” on page 28](#)

Read this topic to understand the concepts behind RESTful web services.

### Related information

[“Planning a JSON service provider application” on page 30](#)

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.

---

## Chapter 2. Configuring web services in CICS

You can configure CICS to support web services, where CICS applications can become web service requesters or service providers. CICS supports different web service specifications, including binary attachments and web services addressing. You can also configure CICS to accept web service requests from IBM MQ or HTTP, and retrieve WSDL files from IBM WebSphere Service Registry and Repository (WSRR).

### Configuring your CICS system for web services

---

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

#### Before you begin

1. Ensure that you have installed Language Environment® support for PL/I. For more information, see [Language Environment support for CICS application programming languages](#) and [Installing CICS support for Language Environment](#).
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](#).

### 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 pickup 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](#).

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](#).

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](#).

## 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 <a href="#">1</a> )	Yes (but see note <a href="#">1</a> )	See note <a href="#">2</a>
Provider	No	Yes	No	Yes	See note <a href="#">2</a>
Requester	Yes	Yes	Yes	See note <a href="#">3</a>	No

Service requester or provider	CICS web services assistant used	PIPELINE required	WEBSERVICE required	URIMAP required	TCPIP SERVICE required
Requester	No	Yes	No	See note <a href="#">3</a>	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 TCPIP SERVICE resource is required when the HTTP transport is used. When the IBM MQ transport is used, a TCPIP SERVICE 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](#).

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.

Web services file or resource	Associated resources
Pipeline configuration file	<ul style="list-style-type: none"> <li>• More than one PIPELINE resource that refers to the file.</li> </ul>
PIPELINE	<ul style="list-style-type: none"> <li>• More than one URIMAP resource that refers to the PIPELINE resource.</li> <li>• More than one WEBSERVICE resource that refers to the PIPELINE resource.</li> <li>• More than one web service binding file in the pickup directory of the PIPELINE resource.</li> </ul>
Web service binding file	<ul style="list-style-type: none"> <li>• 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.</li> <li>• One WEBSERVICE resource that is automatically generated from the binding file. You can define further WEBSERVICE resources if you need to.</li> </ul>
WEBSERVICE	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

*Table 1. Other CICS resources that are associated with each web services file and resource (continued)*

Web services file or resource	Associated resources
URIMAP	<ul style="list-style-type: none"> <li>Just one TCPIPSERVICE resource when it is explicitly named in the URIMAP resource.</li> </ul>
TCPIPSERVICE	<ul style="list-style-type: none"> <li>Many URIMAP resources.</li> </ul>

## 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 IBM MQ transport for SOAP web services

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

**Note:** You cannot use the IBM MQ transport for JSON web services.

### Before you begin

To use the IBM MQ transport with SOAP web services in CICS, you must set up the CICS-MQ adapter in your CICS region. For instructions, see [Setting up the CICS-MQ adapter](#).

You must also specify the CICS system initialization parameter **MQCONN=YES** to start the CICS-MQ connection automatically at CICS initialization. For details, see [MQCONN system initialization parameter](#).

### Overview

#### CICS as a service provider

As a **service provider**, CICS uses IBM 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 \(CKTI\)](#).

You can configure CICS, so that when a message arrives on a local queue, the trigger monitor (provided by IBM 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 IBM 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.

### CICS as a service requester

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 IBM MQ require configuration to define the required resources and queues.

## Defining local queues in a service provider

To use the IBM 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 QNAME attribute of the installed MQMONITOR resource definition for the CICS region, or the value specified for the INITQNAME attribute of the installed MQCONN resource definition.

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('requeuname')
```

where:

- *queuname* 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 QNAME attribute of the installed MQMONITOR resource 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.
  - *requeuname* 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.

## Defining local queues in a service requester

When you use the IBM 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)
```

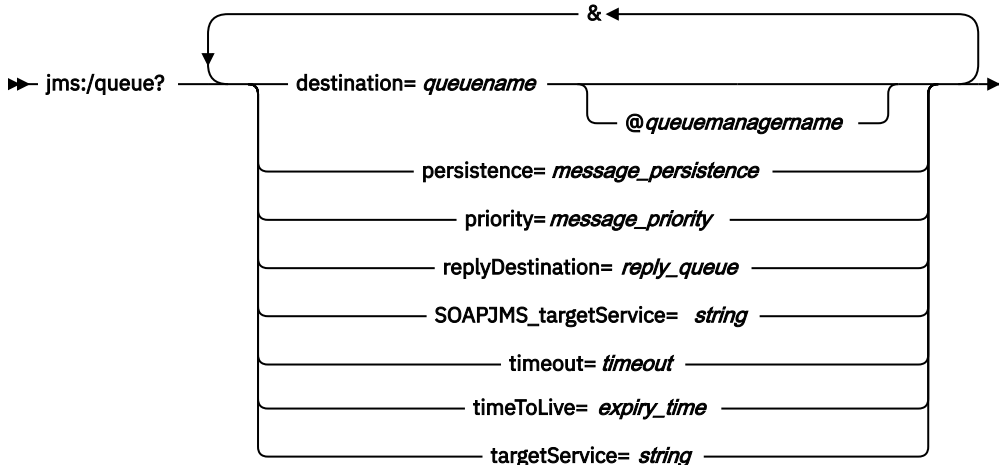
where:

*reply\_queue* is the local queue name.  
*nnn* is the number of retries that will be attempted.

## The URI for the IBM MQ transport

When communication between the service requester and service provider uses IBM 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 IBM MQ.

### Syntax



### Options

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=queueename [ @queuemanagemame ]**

*queueename* is the name of the input queue in the target queue manager  
*queuemanagemame* 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.

#### **SOAPJMS\_targetService=string**

Identifies the target service. If CICS is the service provider, the target service should be of the form '/string', as CICS uses this as the path when attempting to match with URIMAP. If this option is

not specified, the value that is specified in TRIGDATA on the input queue at the service provider is used.

**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 this option is not specified, the value that is specified in TRIGDATA on the input queue at the service provider is used.

**Example**

This example shows a URI for the IBM MQ transport:

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

For information about "connectionFactory" and "initialContextFactory", see the [IBM MQ product documentation](#).

## Configuring CICS to support persistent messages

CICS provides support for sending persistent messages using the IBM 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, follow these steps.

### Procedure

1. Use IDCAMS to define the local request queue and repository file to z/OS.  
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 37](#). 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.

### 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” on page 41](#) for details.

## Persistent message processing

When a web service request is received in an IBM 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 IBM MQ, which might contain RFH2 headers.

The DFHMQMSG data-container contains the IBM 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 IBM 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 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 42.

## 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.

## 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 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.

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. The configuration of a pipeline is specified in an XML document, known as a *pipeline configuration file*, and its name is specified in the CONFIGFILE attribute of a PIPELINE resource definition. For reference information about pipeline configuration elements, see [XML elements for web services pipeline configuration](#).

A pipeline links to message handlers and to header processing programs, if they are used, by using a channel that has a number of containers. CICS uses containers to pass information to and from these programs. The programs also use containers to communicate with other programs in the pipeline. For details about the containers used in web service pipelines, see [Containers used in web services pipelines](#).

## 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 IBM z Systems® 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 18 on page 44 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.

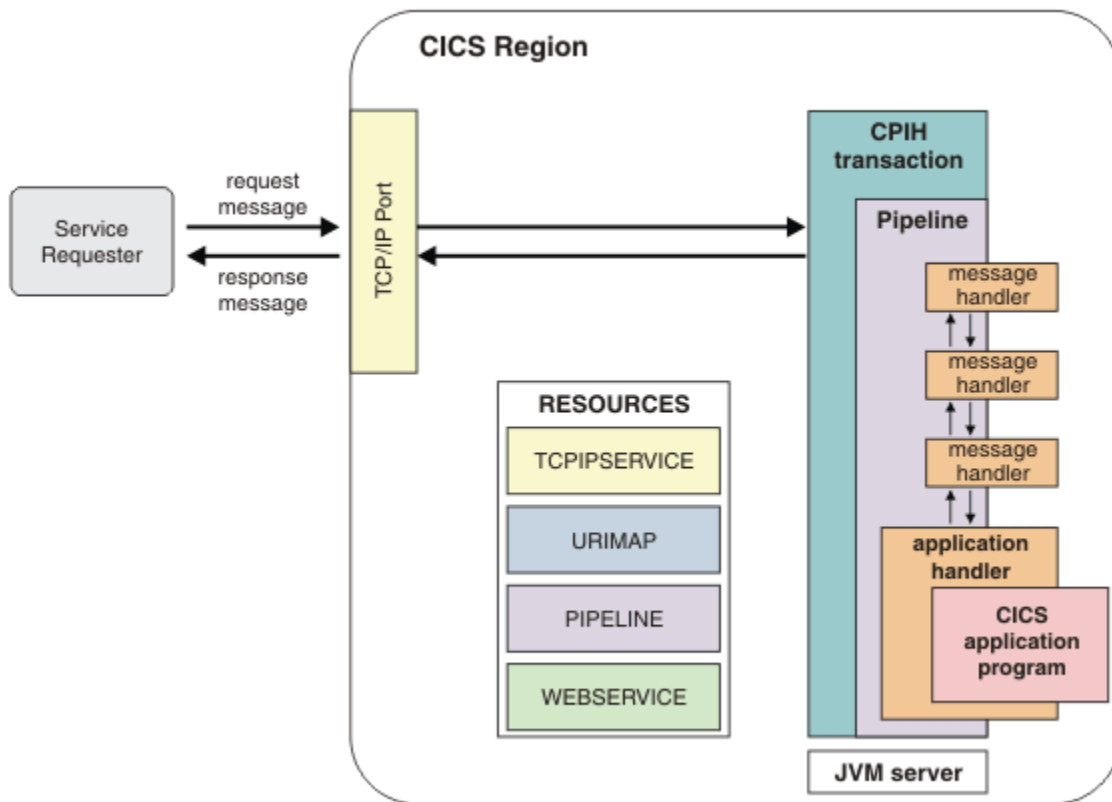


Figure 18. The architecture and resources for a service provider

## How CICS processes a request

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 by using the IBM Z Application Assist Processor (zAAP) if the pipeline is configured appropriately. For more information, see [“Java-based SOAP pipelines”](#) on page 46.

## 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 19 on page 45 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.

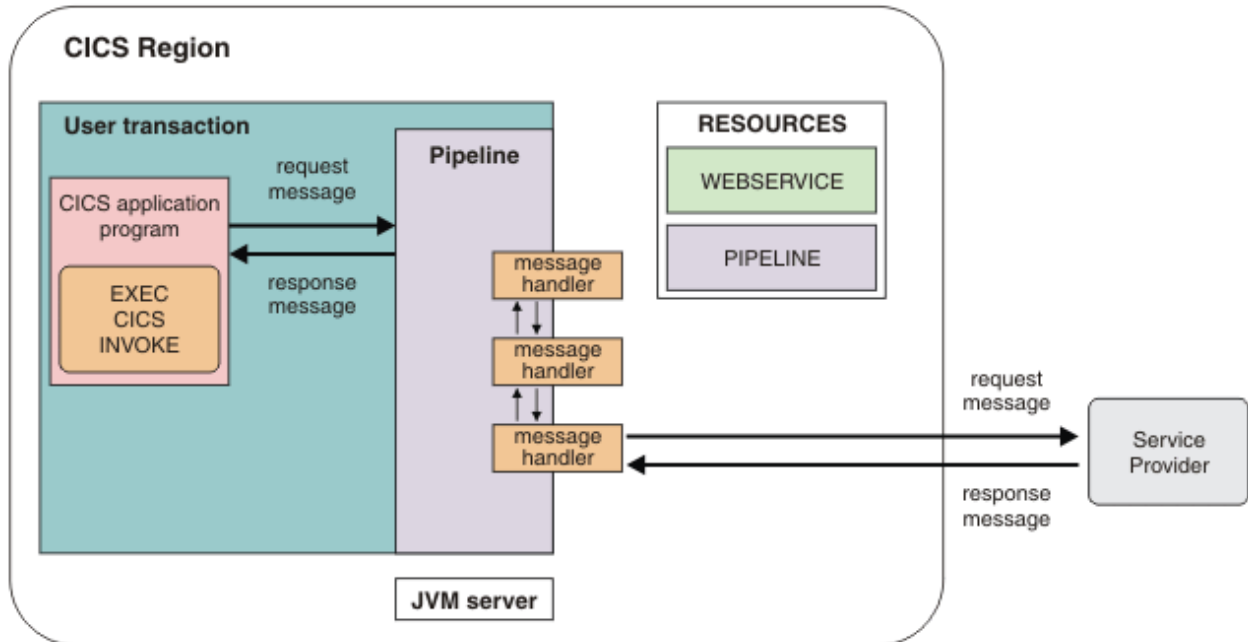


Figure 19. The architecture and resources for a service requester

## How CICS processes a request

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 IBM 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 by using the IBM Z Application Assist Processor (zAAP) if the pipeline is configured appropriately. For more information, see [“Java-based SOAP pipelines”](#) on page 46.

## Java-based SOAP pipelines

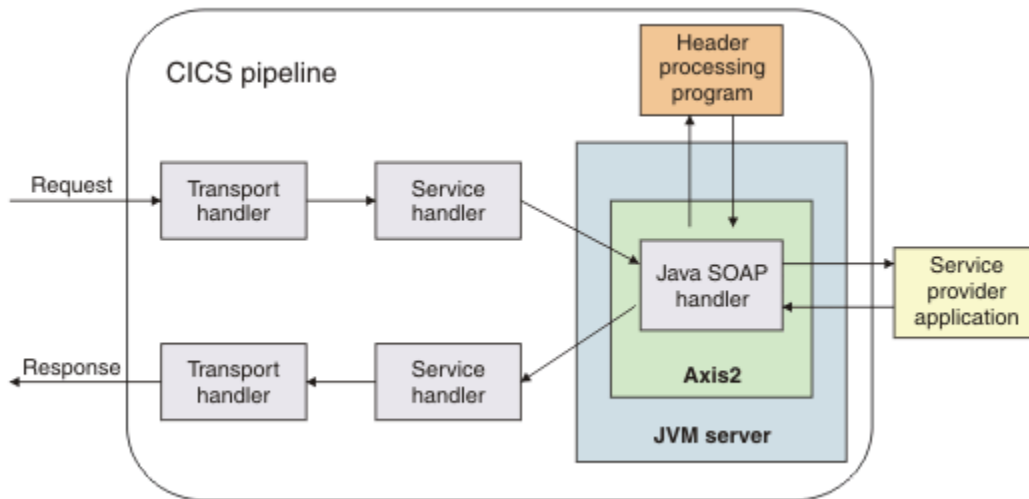
**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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 IBM Z 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 [CICS and Java](#).

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 70.

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 [CICS and Java](#).

## 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 requested 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.

### What options are available

There are several ways of configuring CICS as a service provider for JSON requests:

- Using z/OS Connect

z/OS Connect is a separately-orderable IBM product for accessing z/OS assets, such as CICS programs, by using JSON. For more information about z/OS Connect and how to use it for CICS, see [z/OS Connect Enterprise Edition product documentation](#).

- 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](#).

- Using the JAX-RS and JSON features of the CICS Liberty JVM server directly
- Using the CICS-supplied terminal handler DFHPIJT

You can configure a provider pipeline with terminal handler DFHPIJT. This enables the pipeline to process JSON requests without the necessity of installing a JVM server. The following restrictions apply.

#### Restriction:

- RESTful JSON web services are not supported.
- Context switch in the pipeline is not supported.
- It is not possible to use SOAP and JSON web services in a JSON pipeline. DFHPIJT only handles JSON messages. Receipt of a SOAP message results in an error response.

### Parsing of messages

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 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 IBM Z Application Assist Processor (zAAP) if it is available. Offloading the processing might have cost benefits.
- If you use the CICS-supplied terminal handler DFHPIJT, some workloads might get performance and throughput benefits. When you use this method of processing JSON requests, none of the processing is eligible for offloading onto zAAP.

## CICS as a service provider for JSON requests using CICS Java pipelines

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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 20 on page 49 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](#).

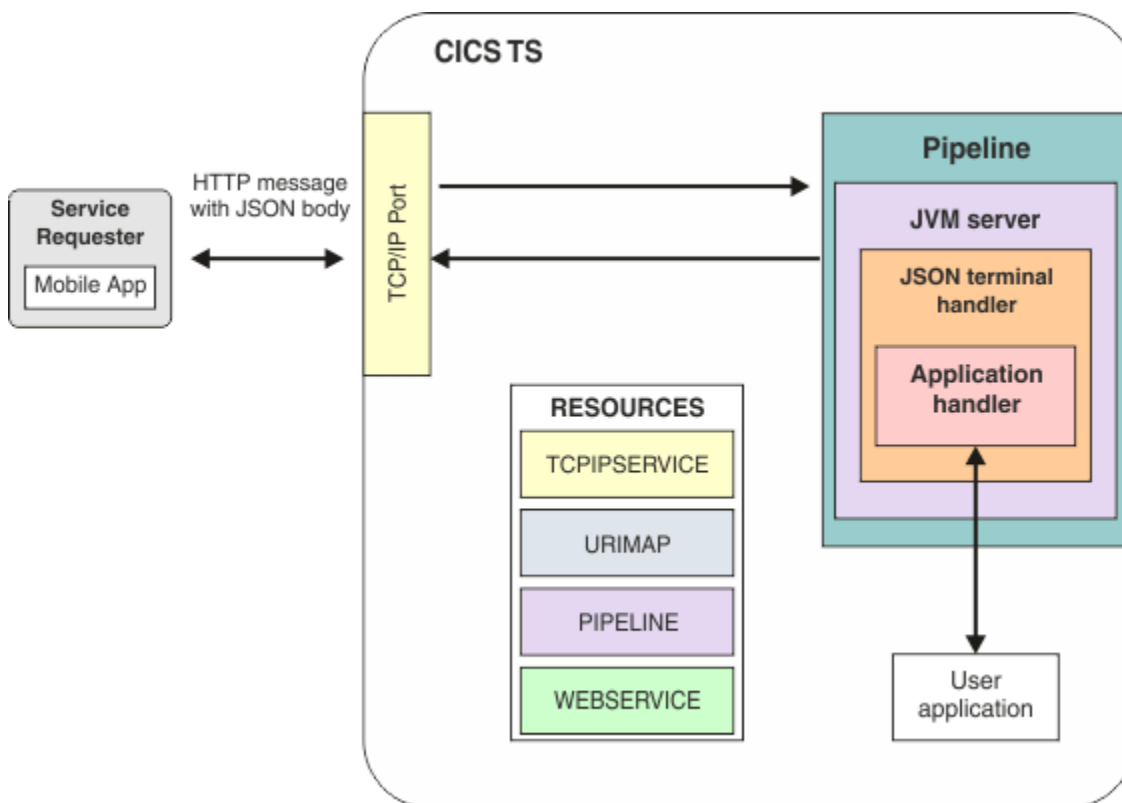


Figure 20. The architecture and resources for a JSON service provider that uses a CICS Java pipeline

### How CICS processes a JSON request

To process a request, CICS completes the following operations:

1. Receive the request from the service requester.

The TCPIP SERVICE 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 IBM Z Application Assist Processor (zAAP).

## Routing provider mode CICS web services

You would like to route provider mode CICS web services requests between CICS terminal owning regions (TORs) and application owning regions (AORs). You might also want to run the workload under a different transaction or user ID than the default. For the configuration of this environment, you would like to know what options are available. You do not want your application programs to run in the TOR.

There are two opportunities to route workload during the processing of a provider mode CICS web services request. However, there are certain limitations and provisos that must be considered. There are also two opportunities to set the transaction identifier that the workload runs under.

There are several stages to the processing of a provider mode web services request, each of which offers certain opportunities to change the transaction and the user ID under which the workload runs, or allows routing decisions to be made. These processing stages can be loosely thought of as being associated with the following resources in CICS:

- The URIMAP
- The PIPELINE
- The WEBSERVICE

You can use any combination of the described techniques to achieve your desired behavior.

### The URIMAP

The [URIMAP resource](#) is used to associate specific web services with their URIs. When a SOAP message arrives in CICS, a URIMAP lookup is performed. If a URIMAP is found that matches the URI from the

incoming SOAP message, then CICS will use this resource to determine which PIPELINE to use to process the SOAP message.

Two significant fields that exist in the URIMAP are the **TRANSACTION** and the **USERID** fields. These fields specify the transaction ID to run the PIPELINE processing under and the user ID to use when attaching this transaction. There are subsequent opportunities to change both of these prior to the application running. If you do not specify these values, CICS will default to using CPIH as the transaction ID (assuming an HTTP based transport mechanism) and will use the region's default user ID. If the SOAP message arrives in CICS over an MQSeries® transport, then the default transaction ID is CPIQ. This transaction is always started in the same region as the URIMAP processing and cannot be routed to a different CICS region.

**Note:** If you use Transport Layer Security (TLS, also referred to as SSL) to sign and encrypt your SOAP messages, then the request might already be being processed under a user ID other than the default. A user ID might be associated with the certificate used to sign the SOAP message, in which case this alternative user ID will already be in effect.

The URIMAP allows you to specify the transaction ID and user ID statically based on the URI. If you need the ability to set them dynamically based on the content of the SOAP messages, review the PIPELINE section below.

If you create your URIMAP resources automatically during a PIPELINE scan operation, you can specify the values to be used for the **TRANSACTION** and **USERID** URIMAP fields when you create your WSBind files. Parameters for this purpose apply to DFHWS2LS and DFHLS2WS called TRANSACTION and USERID. If you are able to pick values for these parameters at the time that the WSBind file is created, it is worth specifying them. If you are unable to do this, you might decide to create URIMAP resources manually or to rely on the PIPELINE processing described below.

## The PIPELINE

The processing that occurs within the PIPELINE can be both sophisticated and complex. For most uses, it is easier to route workload during the WEBSERVICE processing. You could consider skipping this section for now and returning if the WEBSERVICE processing information below is insufficient for your requirements.

The transaction that performs the PIPELINE processing (usually CPIH) is attached locally in the same region that performed the URIMAP processing as there is an affinity between the PIPELINE and the transport mechanism. The transaction is even attached locally if the associated resource definition is defined as either Remote or Dynamic.

With the exception of the Application Handler program, any PIPELINE handler programs that are invoked during the PIPELINE processing will also run locally regardless of the program definitions you have supplied.

The Application Handler program interacts directly with the application program. Both static and dynamic routing of the Application Handler program are possible. Dynamic routing occurs if MRO connections are used and a context switch occurs. When a context switch occurs under certain conditions (see [context switch](#) for details), a separate task is started under a new Unit of Work (UOW), which drives the CICS Distributed Routing user-exit allowing for customized routing decisions to be made. You might use CICSplex® SM to make the routing and work-load balancing decisions for you.

You might specify an alternative user ID and transaction to process the Application Handler program than those currently in effect. This is done by populating either the DFHWS-TRANID container or the DFHWS-USERID container. Any handler program that runs as part of the PIPELINE can change one or both of these values. The handler programs have access to the entire contents of the SOAP messages and can make their decisions on a per-message basis. For example, the CICS supplied WS-Security handler program can place a value in the DFHWS-USERID container if WS-Security is in use.

## The WEBSERVICE

When a provider mode WSBind file is created using the [web services assistants](#), the PGMNAME parameter must be specified. This parameter identifies the name of the application PROGRAM that implements the

web service. CICS performs a Distributed Program Link (DPL) to this PROGRAM after converting the SOAP message into data structures suitable for the application to use. The application program can be hosted in another CICS region such as an AOR.

The DPL causes the CICS Dynamic Routing user-exit to be driven and allows customized routing decisions to be made. You can use CICSplex SM (CPSM) to make the routing and work-load balancing decisions for you. Both static and dynamic routing are possible and normal DPL considerations do apply.

The part of CICS that performs this DPL is the CICS supplied Application Handler called DFHPITP. DFHPITP runs under the same Userid and Transaction that were in effect after the PIPELINE processing completed. By default the transaction identifier will still be CPIH, unless you have changed it using one of the mechanisms described above. Your application program will run under the same transaction and user ID that DFHPITP is using.

Details on the CICS resource definitions required for the DPL routing model and methods of specifying the name of the remote transaction ID for a DPL can be found in section "DPL routing" of [IBM Redbooks: CICS Web Services Workload Management and Availability \(SG24-7144-01\)](#).

In most scenarios, it is simpler to use this DPL to implement routing of provider mode web services than to make changes during the PIPELINE processing as described above.

If you do not use WEBSERVICE resources to implement web services in CICS, you will not be using DFHPITP as the Application Handler, in which case your own Application Handlers can make use of DPLs if you want them to do so.

## Context switch within a service provider pipeline

When a service provider pipeline runs, it is possible for a context switch to occur. This happens only when the terminal handler (`<terminal_handler>`) is one of the CICS-provided SOAP handlers. The context switch causes the program named in the `<apphandler>` element to run in a separate task.

A context switch occurs if any of the following conditions are met:

- The user ID is changed by supplying a new value in the DFHWS-USERID container. Note that the user ID could be changed by the CICS `<wsse_handler>` if WS-Security is being used.
- The transaction ID is changed by supplying a new value in the DFHWS-TRANID container.
- The web service request requires use of Web Services Atomic Transactions (WS-AT).

The context switched task can run locally in the same region or remotely over an MRO connection. The TRANSACTION definition for the context switched task must exist. CICS always ensures that the correct program is executed for the context switched task so the PROGRAM attribute can be set to XXXXXXXX. This allows the same transaction id to be used for multiple different web service requests if needed.

## Considerations for running the context switched task locally

When the context switched task runs locally, consider the following:

- Each web service request requires two tasks, the initial pipeline task and the context switched task. You need to ensure that the **MXT** is sized appropriately to allow for the correct number of tasks to run based on the expected number of concurrent requests. Note that while the context switched task runs, the initial pipeline task will be suspended on resource PIIS RZCBNOTI. For example, to process 50 concurrent requests, you need to set the **MXT** to at least 100.
- The TCLASS is a useful mechanism to prevent the CICS region from being flooded by web service requests. If a TCLASS is used, when a context switch occurs, it is essential that the transaction ID is changed. If the transaction ID is not changed, it's possible that a flood of requests might fill the available TCLASS slots with initial pipeline tasks. When this occurs, the context switched tasks will be suspended waiting for a TCLASS slot to run. This has the effect of deadlocking the system. Note that only the initial pipeline tasks need to be in a TCLASS. The context switched tasks are naturally limited by the number of initial pipeline tasks that can run.
- When the web service uses WS-AT, it is not sufficient to put only the initial pipeline tasks in a TCLASS. This is because the WS-AT context switched task remains in the system after the initial pipeline tasks

have sent the application response back to the requester. It is reasonable to use a separate TCLASS for the WS-AT context switched tasks to avoid them using all the MXT slots while they wait for messages from the WS-AT coordinator. The TCLASS for the WS-AT context switched tasks should have a higher **MAXACTIVE** value than the TCLASS used for the initial pipeline tasks. WS-AT protocol messages are handled by using the DFHRSURI URIMAP. By default, this uses transaction CPIW, which should not be part of a TCLASS.

- Set a suitable DTIMOUT value for the initial pipeline tasks. This ensures that the task eventually times out if the context switched task is not responsive or leaves the system unexpectedly.

## Considerations for running the context switched task remotely

When the context switched task runs remotely, consider the following:

- You will need to install the PIPELINE and WEBSERVICE resources in the target remote region.
- When the context switched task runs in a remote region the origin data for the initial pipeline task is not sent to that region.

## Changing the transaction ID of the context switched task

If you need to change the transaction ID of the context switched task, you have to use a PIPELINE handler program or a SOAP header handler program to do that. The PIPELINE configuration file needs to be updated with the details of the handler program.

Here is an example PIPELINE handler program to change the transaction ID.

```
*ASM      XOPTS(CICS,SP)
DFHEISTG  DSECT
#TRAN     DS      CL4
#LEN      DS      F
TRNHND    DFHEIENT
TRNHND    RMODE ANY
TRNHND    AMODE 31
          MVC      #TRAN,=CL4'CTXT'
          EXEC CICS GET CONTAINER('DFHFUNCTION') SET(6) FLENGTH(#LEN)
          CLC      0(15,6),=C'RECEIVE-REQUEST'
          BNE      RETURN
          EXEC CICS PUT CONTAINER('DFHWS-TRANID') FROM(#TRAN) FLENGTH(4)
          EXEC CICS DELETE CONTAINER('DFHRESPONSE')
RETURN    DS      0H
          EXEC CICS RETURN
END
```

## 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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

### About this task

You can define the PIPELINE resource in a local CICS region using CICS or CICSplex SM functions, or you can use CICS Explorer to define the PIPELINE resource either in a local CICS region or in a CICS bundle.

When you use 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 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 IBM 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.  
See [Configuring CICS to use the IBM MQ transport](#) 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.  
See [CICS resources for web services](#) 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. [Security for SOAP web services](#), to process secure web service messages.
  - d. [Support for Web Services Transactions](#), 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. If you decide to create custom message handler programs, to optimize performance you must make them threadsafe.  
See the following information:
  - [Message handlers and the message handler interface and “Supplying your own message handlers” on page 72](#)
  - [“Header processing programs” on page 67](#) and [“The header processing program interface” on page 68](#)
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 `/usr/lpp/cicsts/cicstsnn/samples/pipelines` directory, where `/usr/lpp/cicsts/cicstsnn` is the default installation directory for CICS files on z/OS UNIX and `cicstsnn` reflects the release of CICS TS. For example, the directory is `/usr/lpp/cicsts/cicsts64/samples/pipelines` for CICS TS beta.

For more information about options available in the pipeline configuration file, see [Pipeline configuration files](#) and [XML elements for web services pipeline configuration](#).
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” on page 54 through “7” on page 54 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.

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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS web services, [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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 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 CICS Explorer to create and edit resources in CICS bundles, see [Working 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 and the message handler interface” on page 59](#) 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, `basicsoap11requester.xml` and `basicsoap11javarequester.xml`, which you can copy and edit as appropriate. These samples are provided in the `/usr/lpp/cicsts/cicstsnn/samples/pipelines` directory, where `/usr/lpp/cicsts/cicstsnn` is the default installation directory for CICS files on z/OS UNIX and `cicstsnn` reflects the release of CICS TS. For example, the directory is `/usr/lpp/cicsts/cicsts64/samples/pipelines` for CICS TS beta. For more information about options available in the pipeline configuration file, see [“Pipeline configuration files” on page 72](#) and [XML elements for web services pipeline configuration](#).

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” on page 56](#) to [“5” on page 56](#) for each different pipeline configuration that you require.
7. Create a PIPELINE resource. See [PIPELINE resources](#).

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](#)).

8. Unless you use autoinstall PROGRAM definitions, create a PROGRAM resource for each program that runs in the pipeline. See [PROGRAM resources](#).  
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](#)) 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](#).

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.

CICS attempts to locate a matching client URIMAP if one was not specified on the **INVOKE SERVICE** command. CICS searches for a matching URIMAP based on the host, port and path of the current request. This allows system programmers to add capabilities from a URIMAP such as using specific certificates and ciphers without the need to update application code.

## 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

Several different technologies exist for implementing JSON services in CICS. The most feature-rich option is z/OS Connect. This task describes the alternative options.

If you do not want to use z/OS Connect, use CICS as a service provider for JSON requests, or use the linkable interface to transform JSON. To use these methods, define and install a JVMSERVER resource with a JVM profile that specifies the **JAVA\_PIPELINE=YES** option. 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 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, see [Creating the CICS infrastructure for a non-Java JSON service provider](#).

**Restriction:** Context switch in the pipeline is not supported.

### Procedure

1. Define the transport infrastructure.

Define a TCPIP SERVICE resource that defines the port on which inbound requests are received. See [“CICS resources for web services” on page 33](#) 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 and the message handler interface” on page 59](#) 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 `/usr/lpp/cicsts/cicstsnn/samples/pipelines` directory, where `/usr/lpp/cicsts/cicstsnn` is the default installation directory for CICS files on z/OS UNIX and `cicstsnn` reflects the release of CICS TS. For example, the directory is `/usr/lpp/cicsts/cicsts64/samples/pipelines` for CICS TS beta. For more information about options available in the pipeline configuration file, see [“Pipeline configuration files” on page 72](#) and [XML elements for web services pipeline configuration](#).

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.

## Creating the CICS infrastructure for a non-Java JSON service provider

You can set up a non-Java environment for processing JSON requests by configuring a provider pipeline with terminal handler DFHPIJT. To create the CICS infrastructure for a non-Java JSON service provider, you must create a pipeline configuration file and create a number of CICS resources.

**Restriction:** If you use this non-Java JSON pipeline to process JSON requests, the following restrictions apply:

- RESTful JSON web services are not supported.
- Context switch in the pipeline is not supported.
- It is not possible to use SOAP and JSON web services in a JSON pipeline. DFHPIJT only handles JSON messages. Receipt of a SOAP message results in an error response.

## Procedure

1. Define the transport infrastructure.

Define a TCPIP SERVICE resource that defines the port on which inbound requests are received. See [“CICS resources for web services” on page 33](#) 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 and the message handler interface” on page 59](#) 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 that contains your message handlers.

In the configuration file, you must specify terminal handler program DFHPIJT in a `<terminal_handler>` element as shown in [Figure 21 on page 59](#). DFHPIJT is the CICS-supplied JSON handler program that enables non-Java processing of JSON messages.

```

<service>
  <terminal_handler>
    <handler>
      <program>DFHPIJT</program><handler_parameter_list/>
    </handler>
  </terminal_handler>
</service>

```

Figure 21. Specifying terminal handler DFHPIJT for non-Java processing of JSON messages

**Note:** When you use DFHPIJT as the terminal handler, do not define an application handler in the pipeline configuration file, that is, the pipeline configuration file should not contain an `<apphandler>` element. If an application handler is specified, it is not invoked.

For more information about options available in the pipeline configuration file, see [“Pipeline configuration files” on page 72](#) and [XML elements for web services pipeline configuration](#).

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.

## Message handlers and the message handler interface

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 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. For an introduction to message handlers and pipelines and an overview of the operation of a pipeline, see [Message handlers and pipelines](#).

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. Your message handler program can retrieve the request from container DFHREQUEST; it can retrieve the response from container DFHRESPONSE.
- 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. See [“Passing a message to the next message handler in the pipeline”](#) on page 64.

It is the contents of the containers which a message handler returns that determines which message handler is invoked next. See [“How containers DFHFUNTION, DFHREQUEST, and DFHRESPONSE control the pipeline protocols”](#) on page 60.

- In a terminal message handler, call an application program, and generate a response. See [“Working with messages in a terminal message handler”](#) on page 65.
- In the request phase of the pipeline, force a transition to the response phase, by absorbing the request, and generating a response. See [“Handling one way messages in a service requester pipeline”](#) on page 64.
- Handle errors. Message handlers should be designed to handle errors that might occur in the pipeline; see [“Handling errors”](#) on page 65.

**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). See [“CICS-supplied SOAP message handlers”](#) on page 66.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS web services, [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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. For reference information about these containers, see [Control containers](#).

#### 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.

For reference information about these containers, see [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.

## How containers DFHFUNTION, DFHREQUEST, and DFHRESPONSE control the pipeline protocols

The contents of the DFHFUNTION, 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 DFHFUNTION determines which control containers are passed to each message handler.

DFHFUNTION	Context	DFHREQUEST	DFHRESPONSE
RECEIVE-REQUEST	Service provider; request phase	Present (length > 0)	Present (length = 0)

DFHFUNCTION	Context	DFHREQUEST	DFHRESPONSE
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, both DFHREQUEST and DFHRESPONSE containers are present. But your message handler can return only one of the containers, and not both. So 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)

<b>DFHFUNCTION</b>	<b>Context</b>	<b>DFHREQUEST</b>	<b>DFHRESPONSE</b>	<b>Action</b>
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
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)

DFHFUNTION	Context	DFHREQUEST	DFHRESPONSE	Action
HANDLER-ERROR	Service requester or provider; either phase	Not applicable	Absent	Call the same handler with the NO-RESPONSE function

You can find reference information about these containers in [Control containers](#).

## Working with messages in a non-terminal message handler

A typical non-terminal message handler processes a message, and then passes control to another message handler in the pipeline. In a non-terminal message handler, you can work with a request or response, with or without changing it, and pass it on to the next message handler.

**Note:** Although web services typically use SOAP messages which contain XML, your message handlers will work as well with other message formats.

## Determining the nature of the message: a request or a response?

Using the contents of container DFHFUNTION, determine whether the message passed to this message handler is a request or a response. [Table 2 on page 63](#) summaries the values in the DFHFUNTION container that are relevant to non-terminal message handlers. The contents identify where in a pipeline a program is being called. For further reference information, see [DFHFUNTION container](#).

<i>Table 2. Contents of container DFHFUNTION relevant to non-terminal message handlers</i>			
DFHFUNTION	Request or response	Type of message handler	Inbound or outbound
RECEIVE-REQUEST	Request	Non-terminal	Inbound
SEND-RESPONSE	Response	Non-terminal	Outbound
SEND-REQUEST	Request	Non-terminal	Outbound
RECEIVE-RESPONSE	Response	Non-terminal	Inbound

### Tip:

- If DFHFUNTION contains PROCESS-REQUEST, the message handler is a terminal message handler, and this topic does not apply.
- If DFHFUNTION contains HANDLER-ERROR, the handler is being called for error processing, and this topic does not apply.

## Retrieving the message

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.

## Processing the message

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. See [“How containers DFHFUNTION, DFHREQUEST, and DFHRESPONSE control the pipeline protocols”](#) on page 60.

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 process a request or response, with or without changing it, and pass it on to the next message handler in the pipeline.

### 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.

### Result

The message is passed to the next message handler on the pipeline.

## Generating or suppressing responses

When you are processing a request, there are times when you want to generate an immediate response, instead of passing the request to the next message handler in the pipeline. **To force a transition to the response phase of the pipeline**, follow these steps:

1. Delete container DFHREQUEST.
2. Construct your response, and put it in container DFHRESPONSE.

The response is passed to the next message handler on the response phase of the pipeline.

In some situations, you want to absorb a request without sending a response. **To suppress the response**, delete containers DFHREQUEST and 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 are invoked again when the response arrives. Some web services do not send a response, and so you must take special action to indicate that CICS should not wait for a response before invoking the message handlers for a second time.

To do this, ensure that container DFHNORESPONSE is present at the end of pipeline processing in the request phase. Typically, this is done by application level code, because the knowledge of whether a response is expected is lodged in the application:

- For applications deployed with the CICS web services assistant, CICS code will create the container.
- Applications that are not deployed with the CICS web services assistant will typically create the container before invoking the application.

If you create or destroy container DFHNORESPONSE in a message handler, you must be sure that doing so will not disturb the message protocol between the service requester and the provider.

## Working with messages in a terminal message handler

A typical terminal handler processes a request, invokes an application program, and generates a response.

**Note:** Although web services typically use SOAP messages which contain XML, your message handlers will work as well with other message formats

### About this task

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 uses 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 invokes 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.

For detailed information about this container, see [DFHFUNCTION](#) container.

**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; see [Control containers](#) for details about this container. 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.

## CICS-supplied 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. SOAP message handlers, and any header processing programs, are specified in the pipeline configuration file.

Depending on which SOAP version is used and whether a pipeline supports Java, you should use one or more of the following message handlers in the pipeline:

- `<cics_soap_1.1_handler>`
- `<cics_soap_1.2_handler>`
- `<cics_soap_1.1_handler_java>`
- `<cics_soap_1.2_handler_java>`

Typically, you need just one SOAP message 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 message 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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

## What actions the SOAP message handlers perform

- 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 SOAP message handlers construct the complete SOAP message, using the `<Body>` element that your application provides.

If you use SOAP headers in your messages, the SOAP message 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.

## 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, 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.

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.

The header processing programs have a channel interface (see “[The header processing program interface](#)” on page 68); the containers hold information that the header processing program can examine or modify, including the SOAP header block for which the program is invoked and the SOAP message body. 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.

The channel and the containers that the header processing program can use are described in [The header processing program containers](#).

## 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 is 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. 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 header processing programs by using channel DFHHHC-V1. Header processing programs are user-written CICS programs that process SOAP header blocks.

Among all the containers that are passed on the channel, container DFHHEADER is specific to the header processing program interface. When a header processing program is called, DFHHEADER contains the single header block that caused the header processing program to be driven.

Containers DFHWS-XMLNS, DFHWS-BODY, and DFHXMLSS-PARSE are also passed on channel DFHHHC-V1. These containers are available elsewhere in your pipeline, but have specific uses in a header processing program.

- DFHWS-XMLNS contains information about XML namespaces that are declared in the SOAP envelope. The header processing program can use this information to resolve qualified names that it encounters in the header block or to construct qualified names in new or modified header blocks.
- DFHWS-BODY contains the body section of the SOAP envelope. The header processing program can modify the contents.
- DFHXMLSS-PARSE is used when you use either the `<cics_soap_1.1_handler>` or `<cics_soap_1.2_handler>` elements in your pipeline configuration. When the header processing program is called, DFHXMLSS-PARSE contains the XML System Services (XMLSS) records for that header.

For further details about containers DFHHEADER, DFHWS-XMLNS, DFHWS-BODY, and DFHXMLSS-PARSE, see [The header processing program containers](#).

What's more, the header processing program interface passes the control containers, context containers, and user containers on channel DFHHHC-V1. For more information about these containers, see [Containers used in web services pipelines](#).

**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](#).

## 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 [“Routing provider mode CICS web services” on page 50](#).

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.

When 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.

## Application handlers

An *application handler* is a CICS program that the terminal handler of a SOAP service provider pipeline links to at run time. 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.

Application handlers are used in provider mode pipelines in which the terminal handler is one of the CICS-supplied SOAP message handlers, that is, 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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

### How application handlers are attached

An application handler can be attached by CICS in two ways:

- The typical mechanism involves a channel and control containers (hence the term *channel-attached application handlers*).
- The other method involves Java bindings for Axis2.

### Channel-attached application handlers

*Channel-attached application handlers* are application handlers that are attached to CICS using a channel and control containers. 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.
- Your own channel-attached application handler. This application handler can be written in languages other than Java.
- 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 [Interface ApplicationHandler](#).

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 the control containers that can be used in your channel-attached application handler, see [Control containers](#).

```
<?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>
```

For an explanation of this example, see [“Example provider pipeline configuration file \(channel-attached application handler\)”](#) on page 78.

*Figure 22. Configuration file for a simple service provider pipeline that uses a channel-attached application handler*

## Axis2 application handlers

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>`.

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](#).

```

<?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>

```

For an explanation of this example, see [“Example provider pipeline configuration file \(Axis2 application handler\)”](#) on page 78.

*Figure 23. Configuration file for a simple service provider pipeline that uses Axis2 application handler*

## Considerations

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](#).

## Supplying your own message handlers

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. However, there are occasions when you want to perform specialized processing on the messages that flow between a service requester and a service provider, and the CICS-supplied message handlers do not meet your needs. Then, you need to supply your own message handlers to perform your own, specialized, operations on web service requests and responses. This topic outlines the general steps to use your own message handlers in a pipeline.

### 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.

## 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 `/usr/lpp/cicsts/cicstsnn/schemas/pipeline/`

directory, where `/usr/lpp/cicsts/cicstsnn` is the default install directory for CICS files on z/OS UNIX and `cicstsnn` reflects the release of CICS TS. For example, the directory is `/usr/lpp/cicsts/cicsts64/schemas/pipeline/` for CICS TS beta.

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 to configure the pipeline from the file.

## The pipeline definition for a service provider

The root element of the pipeline configuration document is the `<provider_pipeline>` element. The high-level structure of the document is shown in [Figure 24 on page 74](#).

**Note:** 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.

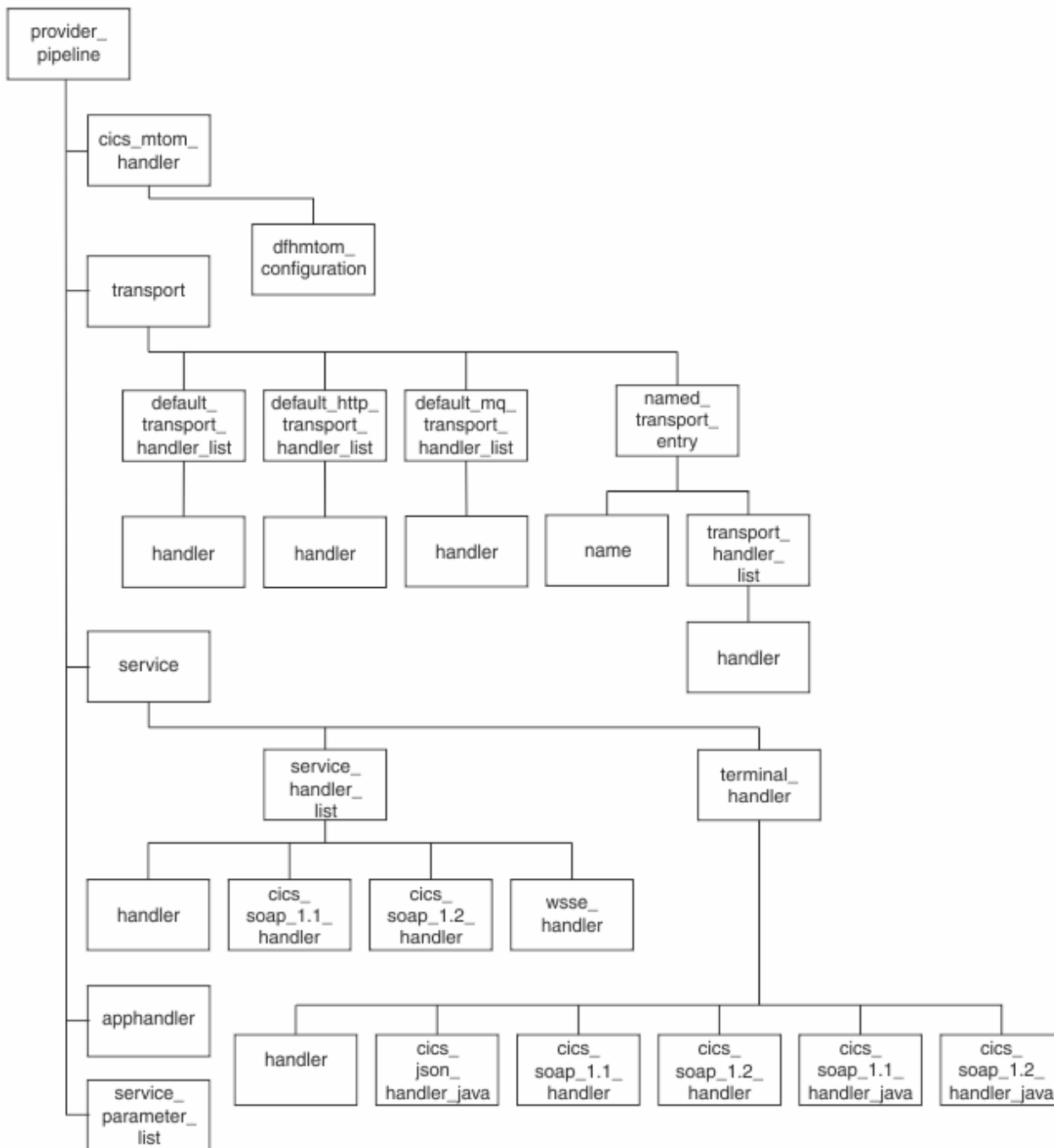


Figure 24. Structure of the pipeline definition for a service provider

## The pipeline definition for a service requester

The root element of the pipeline configuration document is the <requester\_pipeline> element. The high-level structure of the document is shown in Figure 25 on page 75.

**Note:** 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.

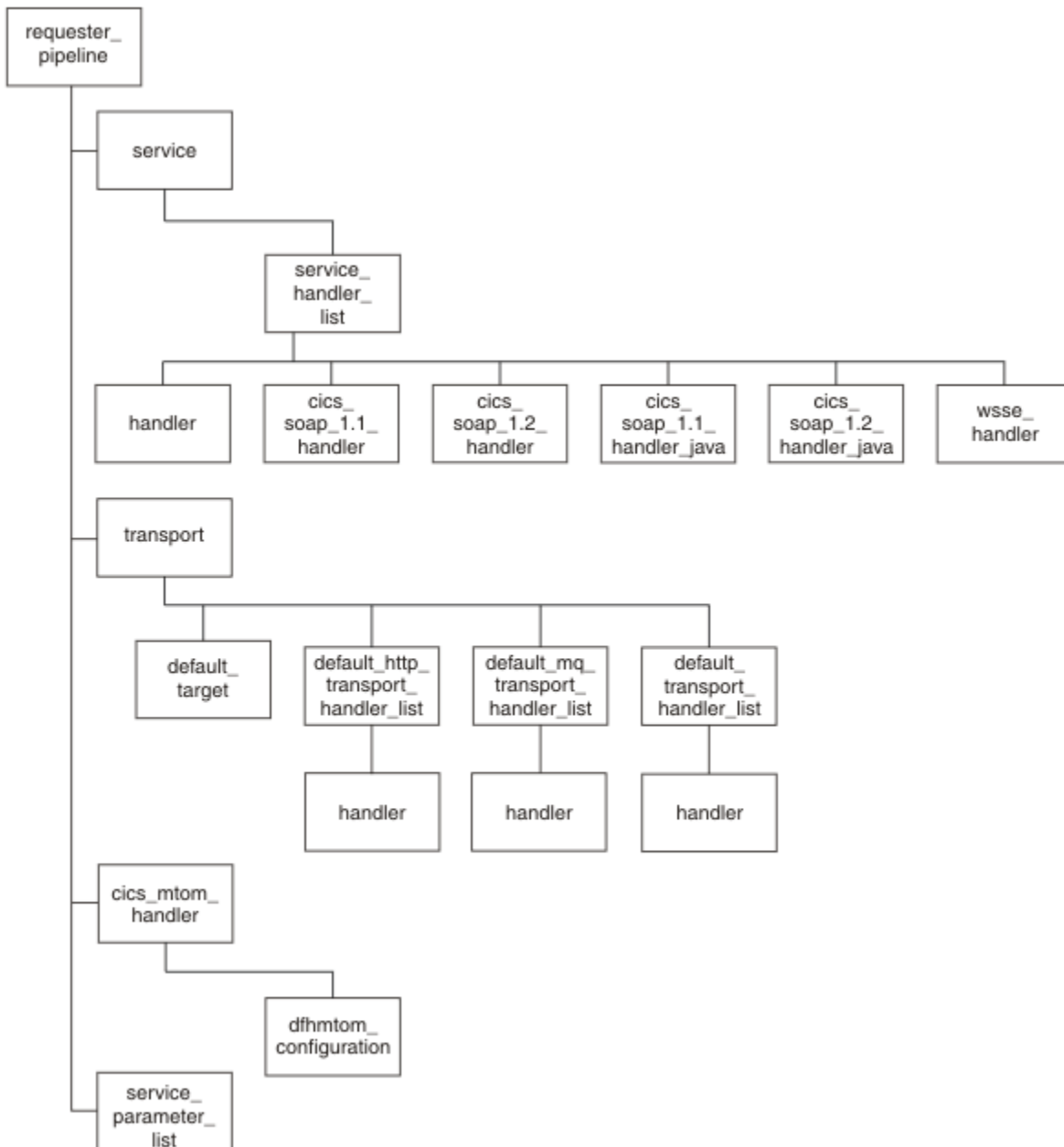


Figure 25. Structure of the pipeline definition for a service requester

## CICS-supplied sample configuration files

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/cicsts/cicstsnn/samples/pipelines` where `cicstsnn` reflects the CICS release; for example, it is `/usr/lpp/cicsts/cicsts64/samples/pipelines` for CICS TS beta.

Copy the sample configuration files to a local directory, and use them as a template to create your own configuration files. The sample files are not intended to be used as is.

**Note:** Ensure that your configuration files are valid in the local CCSID.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java

pipeline for CICS web services, z/OS Connect Enterprise Edition, or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

#### **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**

6.1 6.2

A service provider pipeline definition that adds configuration information for SAML support to `basicsoap11provider.xml`.

6.3 Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

#### **samlrequester.xml**

6.1 6.2

A service requester pipeline definition that adds configuration information for SAML support to `basicsoap11requester.xml`.

6.3 Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

## propagatesamlprovider.xml

6.1 6.2

A service provider pipeline definition that adds configuration information for SAML support with propagation of SAML information through a CICS transaction to `basicsoap11provider.xml`.

6.3 Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

## propagatesamlrequester.xml

6.1 6.2

A service requester pipeline definition that adds configuration information for SAML support with propagation of SAML information through a CICS transaction to `basicsoap11requester.xml`.

6.3 Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

## 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.

## Example provider pipeline configuration file for non-Java JSON parsing

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <handler>
        <program>DFHPIJT</program><handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
</provider_pipeline>
```

```
</terminal_handler>
</service>
</provider_pipeline>
```

The pipeline contains only one message handler.

- 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 only one message handler.
- The `<terminal_handler>` element contains the definition of the terminal message handler of the pipeline.
- The `<handler>` element specifies details of the handler.
- The `<program>` element specifies the program to be invoked. DFHPIJT is the CICS-supplied handler for non-Java JSON processing.

## Specifying transport-related handlers in the pipeline configuration file

CICS supports the use of two transport mechanisms between the web service requester and the provider. 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.

For an introduction to transport-related handlers, see [Transport-related handlers](#).

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 IBM 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 IBM 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 TCPIP SERVICE for the HTTP transport, a QUEUE for the IBM 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 as follows:

### **<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 TCPIP SERVICE for the TCP/IP connection.

### <default\_mq\_transport\_handler\_list>

In a service requester pipeline, the handlers in this list are called when the IBM MQ transport is in use.

In a service provider pipeline, the handlers in this list are called when the IBM 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 IBM MQ transport, the handlers in this list are called when the resource name matches the name of the message queue that receives the inbound message.

### Example

This is an example of a <transport> element from the pipeline configuration file for a service provider pipeline:

```
<transport>
  <!-- HANDLER1 and HANDLER2 are the default transport handlers -->
  <default_transport_handler_list>
    <handler><program>HANDLER1</program><handler_parameter_list/></handler>
    <handler><program>HANDLER2</program><handler_parameter_list/></handler>
  </default_transport_handler_list>

  <!-- HANDLER3 overrides defaults for MQ transport -->
  <default_mq_transport_handler_list>
    <handler><program>HANDLER3</program><handler_parameter_list/></handler>
  </default_mq_transport_handler_list>

  <!-- HANDLER4 overrides defaults for http transport with TCPIPService(WS00) -->
  <named_transport_entry type="http">
    <name>WS00</name>
    <transport_handler_list>
      <handler><program>HANDLER4</program><handler_parameter_list/></handler>
    </transport_handler_list>
  </named_transport_entry>
</transport>
```

The effect of this definition is this:

- The <default\_mq\_transport\_handler\_list> ensures that messages that use the MQ transport are processed by handler HANDLER3.
- The <named\_transport\_entry> ensures that messages that use the TCP/IP connection associated with TCPIPService(WS00) are processed by handler HANDLER4.
- The <default\_transport\_handler\_list> ensures that all remaining messages, that is, those that use the HTTP transport, but not TCPIPService(WS00), are processed by handlers HANDLER1 and HANDLER2.

**Remember:** Any handlers specified in the service section of the pipeline definition are called in addition to those specified in the transport section.

## 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.

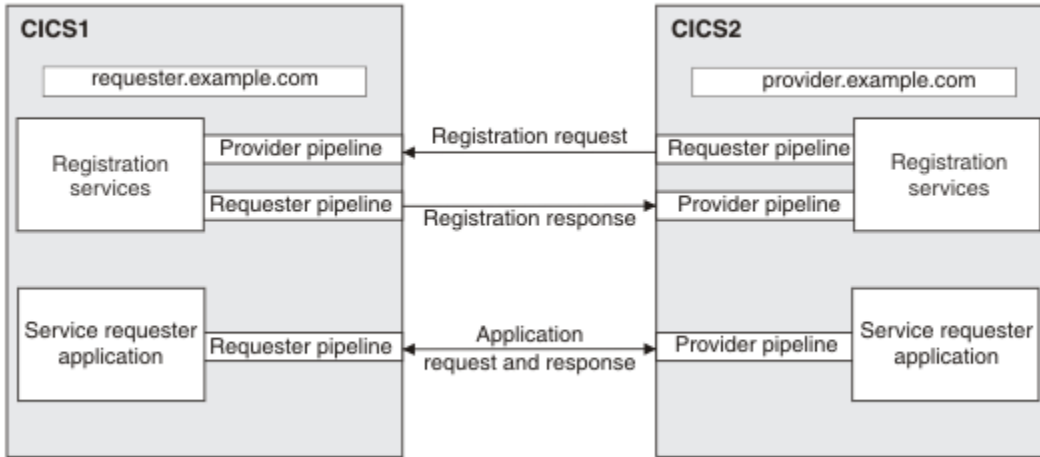


Figure 26. Registration services

Figure 26 on page 82 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 27 on page 83](#).

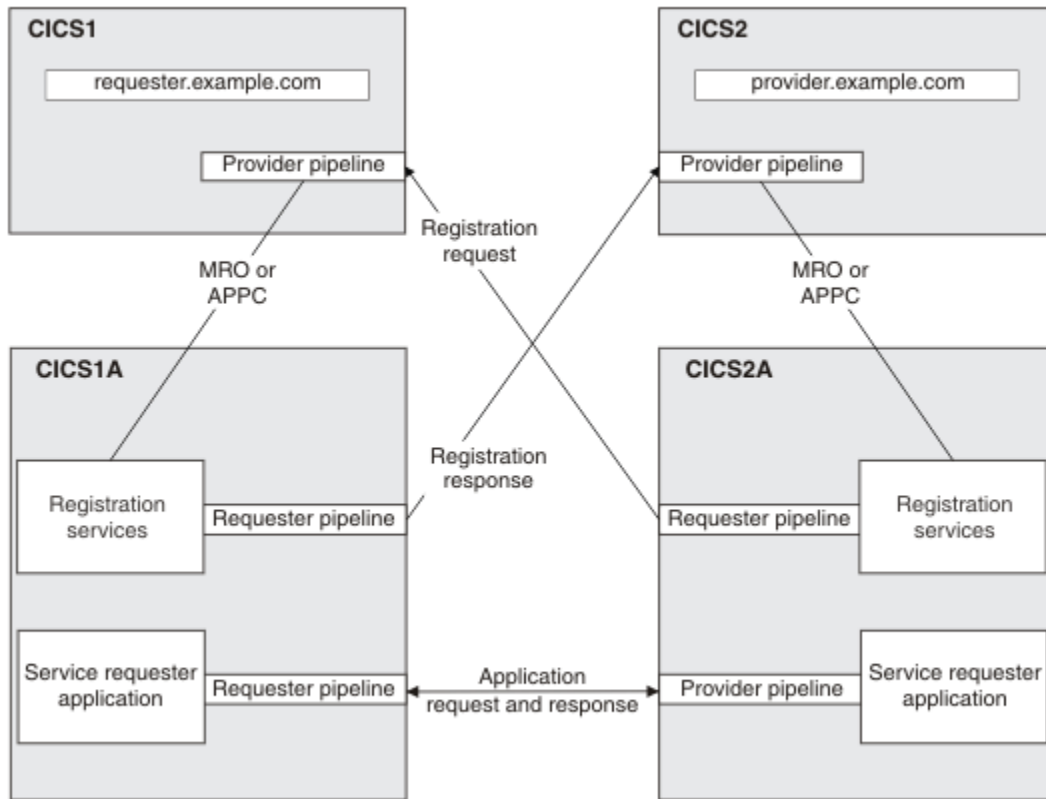


Figure 27. Registration services in a CICSplex

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 27 on page 83](#) 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/cicsts64/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.

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.

## Procedure

To configure CICS for web service transactions:

1. Add the DFHPIDIR data set to your startup JCL.

DFHPIDIR stores a mapping between contexts and tasks.

- a) Add a new DD statement for the DFHPIDIR data set to your CICS startup JCL
- b) Create the DFHPIDIR data set using information in DFHDEFDS.JCL.

The default RECORDSIZE of DFHPIDIR is 1 KB, which is adequate for most uses. You can create DFHPIDIR with a larger RECORDSIZE if you need to.

- c) Install the appropriate group for the data set on your CICS system: DFHPIVS, DFHPIVR, or DFHPICF.

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/cicsts64/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/cicsts64/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.

## 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/cicstsn/samples/pipelines/wsatprovider.xml`, where `/usr/lpp/cicsts/cicstsn/samples/pipelines/wsatprovider.xml` is the default installation directory for CICS files on z/OS UNIX and `nn` reflects the release of CICS TS. For example, `/usr/lpp/cicsts/cicsts64/samples/pipelines/wsatprovider.xml` is for CICS TS beta.

## 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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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/cicstsn/samples/pipelines/wsatrequester.xml`, where `/usr/lpp/cicsts/cicstsn/samples/pipelines/wsatrequester.xml` is the default installation directory for CICS files on z/OS UNIX and *nn* reflects the release of CICS TS. For example, `/usr/lpp/cicsts/cicsts64/samples/pipelines/wsatrequester.xml` is for CICS TS beta.

### 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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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/wscoord</namespace>
    <localname>CoordinationContext</localname>
    <mandatory>true</mandatory>
```

```
</headerprogram>
</cics_soap_1.1_handler>
```

You can include other `<headerprogram>` elements if your application needs them.

2. Code a `<registration_service_endpoint>` element in a `<service_parameter_list>`. Code the `<registration_service_endpoint>` as follows:

```
<registration_service_endpoint>
http://address:port/cicswsat/RegistrationService
</registration_service_endpoint>
```

*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.
  - a) Perform an auxiliary trace using component PI and set the tracing level to 2.
  - b) Look for trace point PI OA31, 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.

### Procedure

If you need to access this information, you can follow one of the following steps:

- Look inside the contents of the <cicswsa:Action> element using a trace.
  - a) Perform an auxiliary trace using component PI and set the tracing level to 2.
  - b) Look for trace point PI OA31, 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/wsac/Prepare"  
"http://schemas.xmlsoap.org/ws/2004/10/wsac/Prepared"  
"http://schemas.xmlsoap.org/ws/2004/10/wsac/Commit"  
"http://schemas.xmlsoap.org/ws/2004/10/wsac/Committed "
```

The states for a transaction that is rolled back are:

```
"http://schemas.xmlsoap.org/ws/2004/10/wscor/Register"  
"http://schemas.xmlsoap.org/ws/2004/10/wscor/RegisterResponse"  
"http://schemas.xmlsoap.org/ws/2004/10/wsac/Rollback"  
"http://schemas.xmlsoap.org/ws/2004/10/wsac/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 IBM 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. To support MTOM messages in CICS, you must specify the correct MTOM/XOP support for your type of pipeline in your pipeline configuration files.

### 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 additional pipeline processing is required because the validation processing cannot handle the contents of XOP documents.

## Configuring CICS to support MTOM/XOP for Java-based pipelines

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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 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.

Review [“Restrictions for Java-based pipelines”](#) on page 94.

### 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 CICS to support MTOM/XOP for other SOAP pipelines

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. This instruction is intended for pipelines that do not use the `<cics_soap_1.1_handler_java>` or `<cics_soap_1.2_handler_java>` handlers. To enable MTOM/XOP support for non-Java pipelines, 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.

Review [“Restrictions for other SOAP pipelines”](#) on page 94.

### 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.
- 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.

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.

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.

## 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.

### 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.

## 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 IBM 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.

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.

Table 3. 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
[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: <a href="http://www.w3.org/2005/08/addressing/anonymous">http://www.w3.org/2005/08/addressing/anonymous</a> .  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.
[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 <a href="http://www.w3.org/2005/08/addressing/anonymous">http://www.w3.org/2005/08/addressing/anonymous</a> .
[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 <a href="http://www.w3.org/2005/08/addressing/reply">http://www.w3.org/2005/08/addressing/reply</a> .
[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://www.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 4. 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/wsdli-instance"
    wsdli:wSDLiLocation="http://example.ibm.com/wsdli/wsdli-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 5. 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

## 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>JVMSEV1</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 JVMSEV1 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

## 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

## 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/wsd1/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
[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

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

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

### Related information

[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.

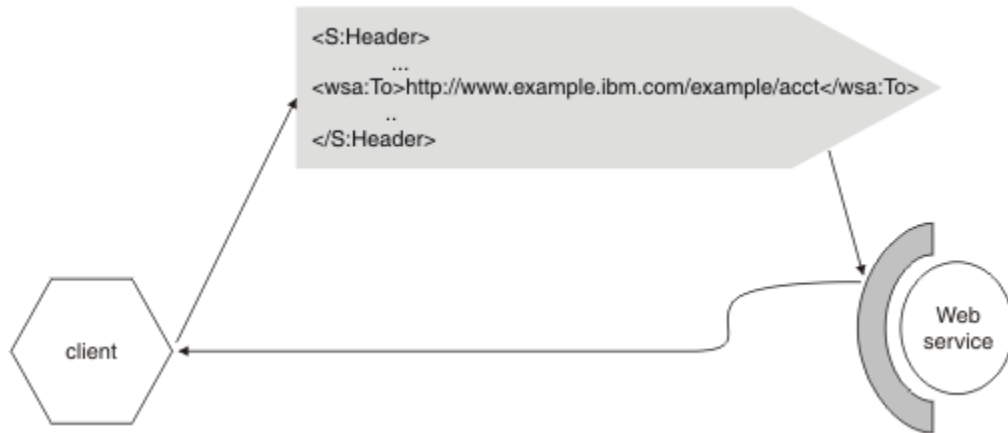
If you are using the recommendation specifications and do not specify a value for the <wsa:ReplyTo> MAP, the <wsa:ReplyTo> MAP defaults to an endpoint reference that contains the anonymous URI (<http://www.w3.org/2005/08/addressing/anonymous>), which causes CICS to send the response back to the requester.

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

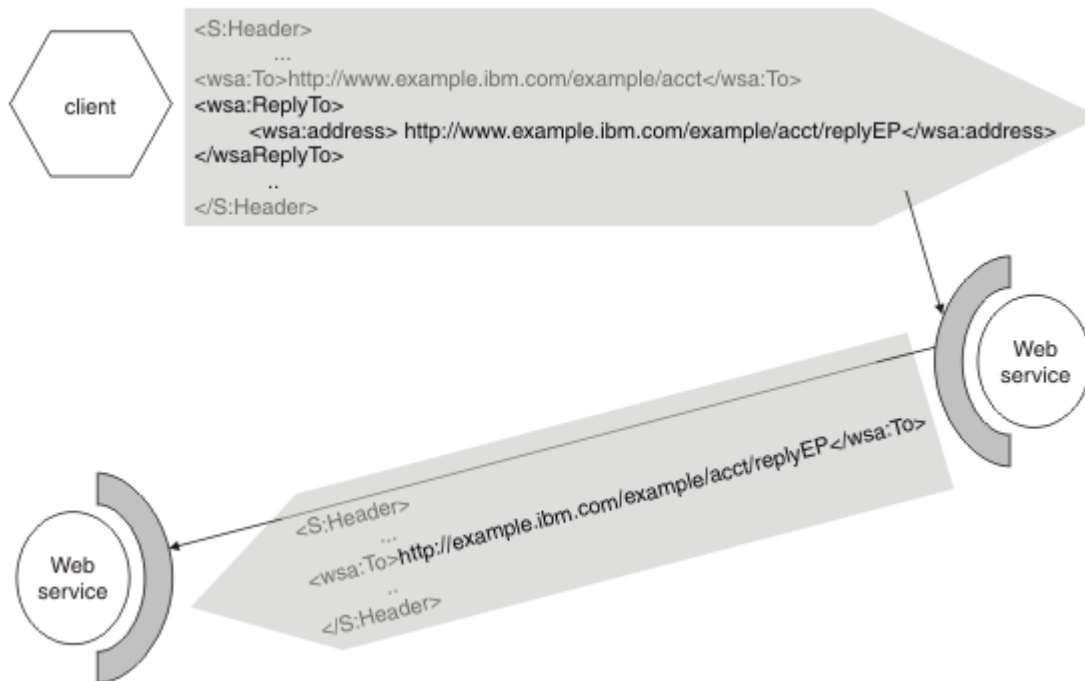
By default, the response part of a two-way message is returned according to the underlying protocol in use. In the case of an HTTP request, the response is returned synchronously in the HTTP response.



### 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 an IBM 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 111.

## 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:

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)

Table 6. Mandatory message addressing properties for WS-Addressing. (continued)

WS-Addressing MAP name	Description	Mandatory in WSDL 1.1	Mandatory in WSDL 2.0
<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](#).
- 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>.

## 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.

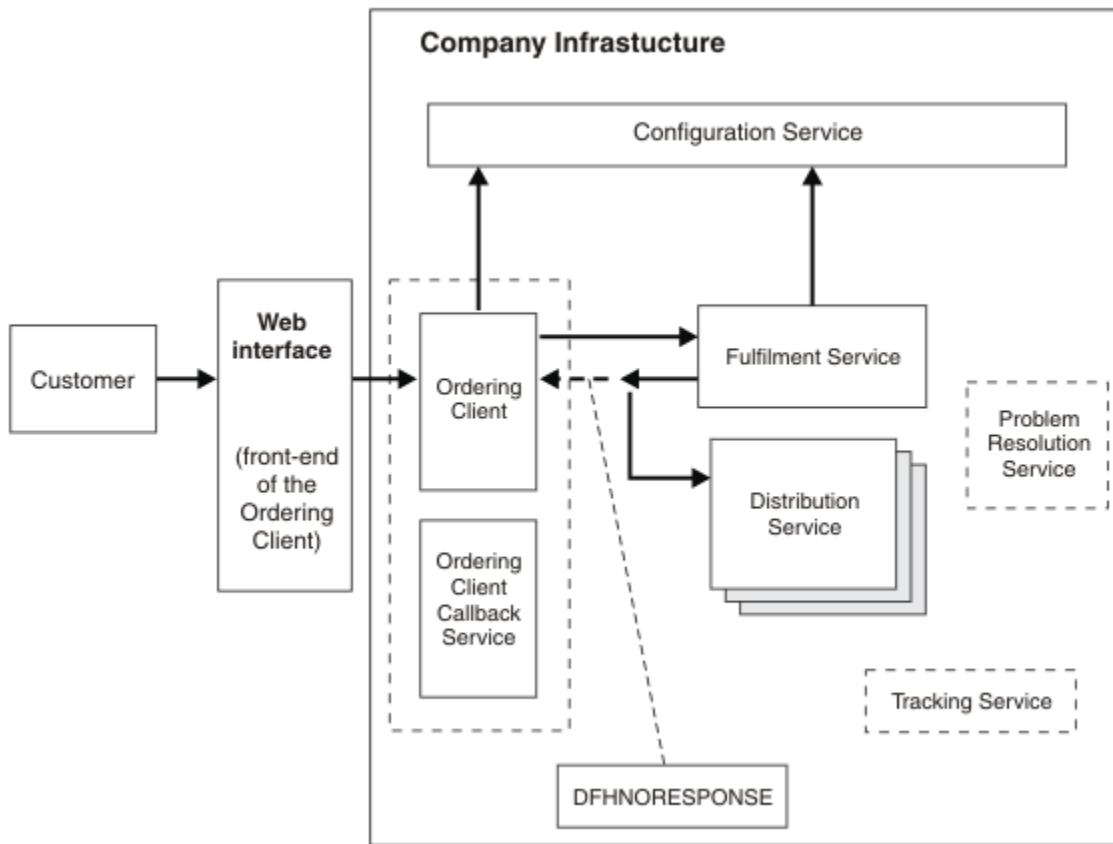


Figure 28. Company infrastructure

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 29 on page 116 shows the request-response message exchange between the Ordering Client and the Configuration Service. This message exchange involves two typical web services pipelines.

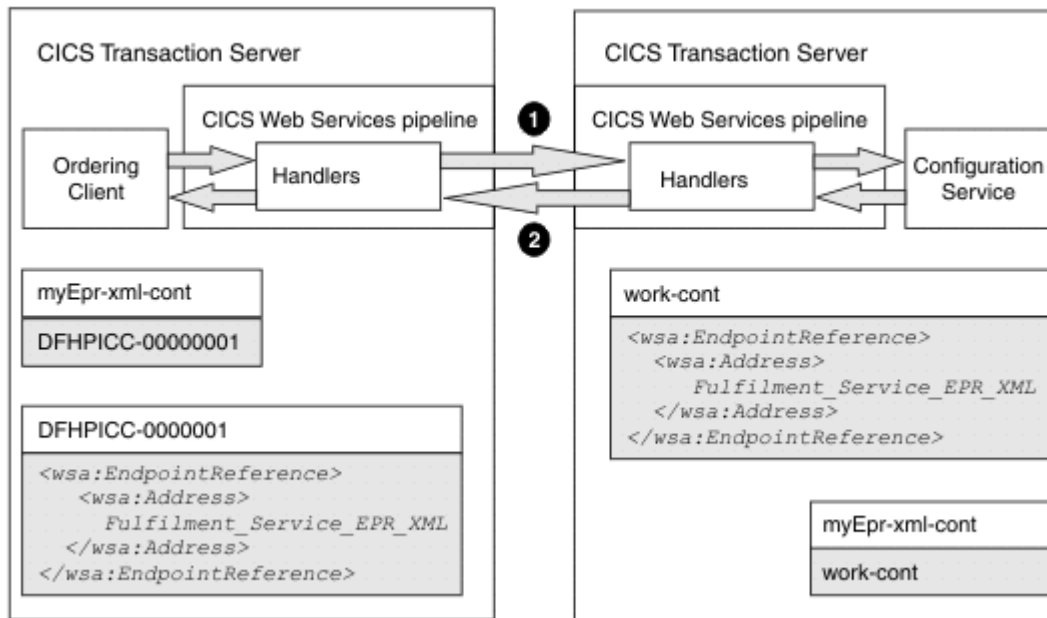


Figure 29. Request-response message exchange between the Ordering Client and the Configuration Service

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 30 on page 117 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.

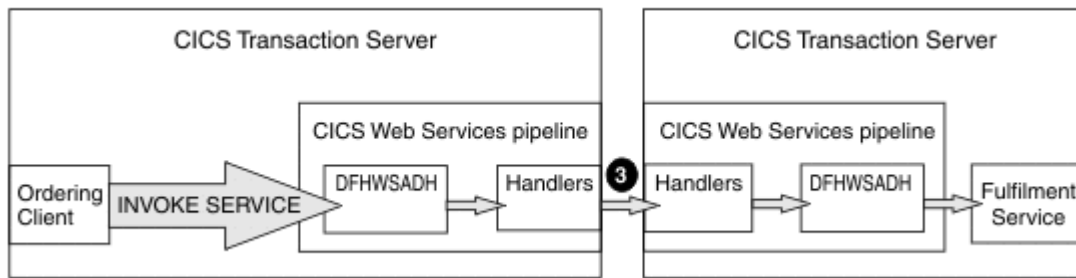


Figure 30. Request from the Ordering Client to the Fulfilment Service

## Support for WS-Security

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

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.

For information about designing and configuring security, see [Security for SOAP web services](#).

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>
```

## Controlling 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 119.

3. If the command was successful, retrieve the response containers from the channel.

At the minimum, container DFHWS-DATA will be present. It holds the top level data structure that CICS created from the SOAP response. If the response contains any arrays that have varying numbers of elements, they are represented as series of connected data structures in a series of containers. These containers will be present in the channel.

4. If the service requester receives a SOAP fault message from the invoked web service, you 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 resolve 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 123. 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 IBM 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
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

CICS also supports the canonical forms of the numeric character references used for white space codes:

Character	XML entity
Tab	&#x9;
Carriage return	&#xA;

Character	XML entity
Line feed	&#xD;

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: &lt; here &gt; ". 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. 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 [Pipeline domain exits](#).

## Procedure

- 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, XWSRROI, 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.
- 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.

## 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

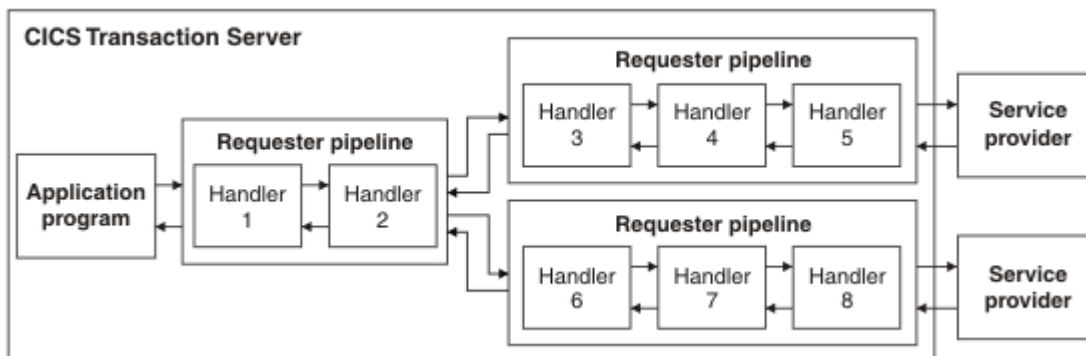
If you change the URI to the format `cics://PROGRAM/program`, where *program* is the name of the target application program, CICS passes the current channel and its containers or COMMAREA to the program using an **EXEC CICS LINK** command.

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.

## 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 123](#) 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 [DFHWS-URI container](#).

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.



---

## Chapter 3. Developing web services

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

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

You can expose existing CICS applications as JSON web services and create CICS applications to act as JSON web service providers. CICS does not provide built in support for requester mode JSON web services. But you can write CICS application programs to call a RESTful web service hosted on another system.

Follow this section to deploy your CICS applications in a web services environment based on the protocol used.

### Before you begin

For fundamentals on CICS support for web services, see [CICS and web services](#).

---

## Developing JSON web services

You can expose existing CICS applications as JSON web services and create CICS applications to act as JSON web service providers. CICS does not provide built in support for requester mode JSON web services. But you can write CICS application programs to call a RESTful web service hosted on another system.

### Before you begin

You must configure your CICS system to support JSON web services. For more information, see [Creating the CICS infrastructure for a JSON service provider](#).

You must have fundamental knowledge about [how CICS supports web services](#). See also [Getting started with JSON web services](#).

Be aware of [“JSON web service restrictions”](#) on page 138.

### Procedure

1. Create a JSON web service.

#### For JSON web service provider applications

You can create a JSON web service provider application in one of three ways:

- Use the [CICS 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.

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

See [“Creating a JSON service provider application”](#) on page 128 for detailed instructions.

- Use z/OS Connect to generate the mapping between a JSON schema or Swagger document and application data structures. See [z/OS Connect Enterprise Edition product documentation](#) to learn more.
- Use your own code to map between JSON and application data structures.

#### **For JSON service requester applications**

You must use z/OS Connect Enterprise Edition for requester mode JSON web services. See [z/OS Connect Enterprise Edition product documentation](#) to learn more.

#### **For CICS applications to invoke remote JSON web services**

You can do so without using z/OS Connect. You can write an application program to call a RESTful web service by using either the **TRANSFORM DATATOJSON** and **TRANSFORM JSONTODATA** API commands or the linkable interface, DFHJSON, to transform JSON and then using the WEB commands to send it to the remote service provider. For detailed instructions, see [“Creating a JSON web service client application”](#) on page 137.

2. Start the JSON web service to test that it works as you intended.

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

## **Creating a JSON service provider application**

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.

### **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**, **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](#).

## 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 134.

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](#).

## What to do next

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

## 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 copybook) 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 131.
- 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 7 on page 132](#) . 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/\*

Container	Contents	Example
	The complete URI	
	The portion of URI path that matched the URIMAP	
	The query string	

- 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](#).
- 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 131.
  - 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 131.
  - 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](#). 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 131.
- The CICS Program from step “2” on page 132.
- 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](#).

## ***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 container](#).

**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.

## 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/cicsts64/schemas/channel directory (where /usr/lpp/cicsts/cicsts64 is the default install directory for CICS files on z/OS UNIX) and cicsts64 represents the CICS version.

### **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>
        <attribute name="name" type="tns:name16Type" use="optional" />
      </sequence>
    </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 136.

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 JSON web service client application

You can write an application program to call a RESTful web service by using either the **TRANSFORM DATATOJSON** and **TRANSFORM JSONTODATA** API commands or the linkable interface, DFHJSON, to transform JSON and then using the WEB commands to send it to the remote service provider.

### Before you begin

You must be familiar with the **EXEC CICS WEB** APIs, as described in [Making HTTP requests through CICS as an HTTP client](#).

If you intend to write applications that use JSON for purposes other than a web services client, see [Mapping and transforming application data and JSON](#).

### About this task

As part of a CICS application you might want to call a RESTful web service hosted on another system. To do this, you must first describe the data to be exchanged with the remote service. You can then write an application program that uses the **EXEC CICS WEB** API to communicate with the remote service with the HTTP protocol to send request data to the service and receive response data. You can use either the **TRANSFORM DATATOJSON** API command or the linkable interface, DFHJSON, to transform your application data to JSON to use as part of the request, and use either **TRANSFORM JSONTODATA** or DFHJSON to transform the JSON response into application data. Some services might not support a payload for both the request and response.

The linkable interface DFHJSON is an older technology, superseded by the **EXEC CICS TRANSFORM** API commands. Although both can be used to convert JSON data to and from copybook-defined application data, you are advised to use the **EXEC CICS TRANSFORM** API commands.

### Procedure

1. Define the interface to the remote service.
  - a) If the remote service exists, check to see whether a JSON schema is available that describes the request and response payloads. If not, you must create one. Then, use the JSON assistant to generate a mapping to a language structure. For more information, see [Generating mappings from a JSON schema](#).
  - b) If the remote service does not yet exist and you want to base its interface on your application's data structure, use the JSON assistant to generate a JSON schema. Then, pass the JSON schema to the remote service application developer. For more information, see [Generating mappings from language structure](#).
2. Define a BUNDLE resource for the bundle that is generated by the JSON assistant and install the bundle in CICS.
3. Define a URIMAP resource for the remote service endpoint and install it.  
For more information, see [URIMAP resources](#).
4. Create or update an application program to call the remote service as follows:
  - a) If the remote service requires a JSON payload for the request (for example when the HTTP method is POST or PUT), transform your application data to JSON.

To use **TRANSFORM DATATOJSON** to process data conversion in your application, see [“Transforming application data to JSON by using the TRANSFORM DATATOJSON API command” on page 305.](#)

To use the linkable interface, DFHJSON, to let CICS or a JVM server to process data conversion, see [Transforming application data to JSON by linking to DFHJSON.](#)

- b) Open a connection to the server where the remote service is hosted, by using the **EXEC CICS WEB OPEN** command.

For more information, see [WEB OPEN](#).

- c) Depending on the requirements of the service, you might want to code the **EXEC CICS WEB WRITE HTTPHEADER** command to specify the Content-Type header application/JSON to indicate that JSON is being supplied.

For more information, see [WEB WRITE HTTPHEADER](#).

- d) Code an **EXEC CICS WEB CONVERSE** command to send the request to the remote service and receive the response. Specify the query string or request body (from the DFHJSON-JSON container) if required. If you expect a response from the service, specify the DFHJSON-JSON container to receive the response JSON.

For more information, see [WEB CONVERSE](#).

- e) If you do not expect to make further requests, code an **EXEC CICS WEB CLOSE** command to close the connection.

For more information, see [WEB CLOSE](#).

- f) Check the HTTP response code that is returned by the **EXEC CICS WEB CONVERSE** command and take appropriate action if an error occurred.

For example, try the request again or return an error to the user.

- g) If a response body was expected from the remote service, transform the JSON response to application data.

To use the **TRANSFORM JSONTODATA** API command, see [“Transforming JSON to application data by using the TRANSFORM JSONTODATA API command” on page 307.](#)

To use the linkable interface, DFHJSON, see [Transforming application data to JSON by linking to DFHJSON.](#)

## Results

You have created an application that can call a RESTful web service with a JSON payload.

## 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.
- IBM 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 mandatory 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 quotation marks.
- Use of simple data types is not supported at the root of a JSON schema. The JSON schema describes a JSON object or a JSON array, although the JSON object can in turn, contain simple data types, arrays, and other objects.
- 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 it generates JSON at run time.

**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:** Some pipeline containers are not populated when a JSON request is processed. For more information, see [Containers used in web services pipelines](#).

## 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.

## Developing SOAP web services

---

You can expose existing CICS applications as SOAP web services and create 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 [Creating the web services infrastructure](#).
3. Decide whether you want to use the web services assistant; see [Planning to use SOAP web services](#).

You must have fundamental knowledge about [how CICS supports web services](#).

## Procedure

1. Create a SOAP web service in one of four ways:

- Use the [CICS 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.

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

Follow the instruction in the following topics:

- [“Creating a web service provider by using the web services assistant” on page 171](#)
- [“Creating a web service requester using the web services assistant” on page 181](#)
- Use IBM Developer for z/OS 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. For details, see [“Developing web services using tooling” on page 183](#).
- 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. For more information, see [“Creating your own XML-aware web service applications” on page 183](#).
- Deploy an Axis2 application as a web service. For details, see [“Using Java with web services” on page 192](#).

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.

## 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 *h1q*.XDFHINST library.

The relevant usage mode for the DFHLS2WS or DFHWS2LS procedure depends on your requirements:

- [“DFHLS2WS: high-level language to WSDL conversion” on page 141](#)
- [“DFHWS2LS: WSDL to high-level language conversion” on page 154](#)

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. For details, see [“Customizing generated web service description documents” on page 177](#).

For information about the data mappings between high-level language structures and XML schemas or WSDL documents, see the following topics:

- [“How the CICS assistants map between high-level languages and XML schemas” on page 329](#)
- [“Mapping levels for the CICS assistants” on page 332](#)
- [“Data mapping limitations when using the CICS assistants” on page 330](#)
- [“Variable arrays of elements” on page 368](#)

## 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

## 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 **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 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.

**PATHMAIN =*path***

Specifies the main part of the name of the CICS TS directory in the UNIX system services file system.

The default value is /usr/lpp/cicsts.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHMAIN** parameter.

**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 the value specified by the **PATHMAIN** parameter.

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:

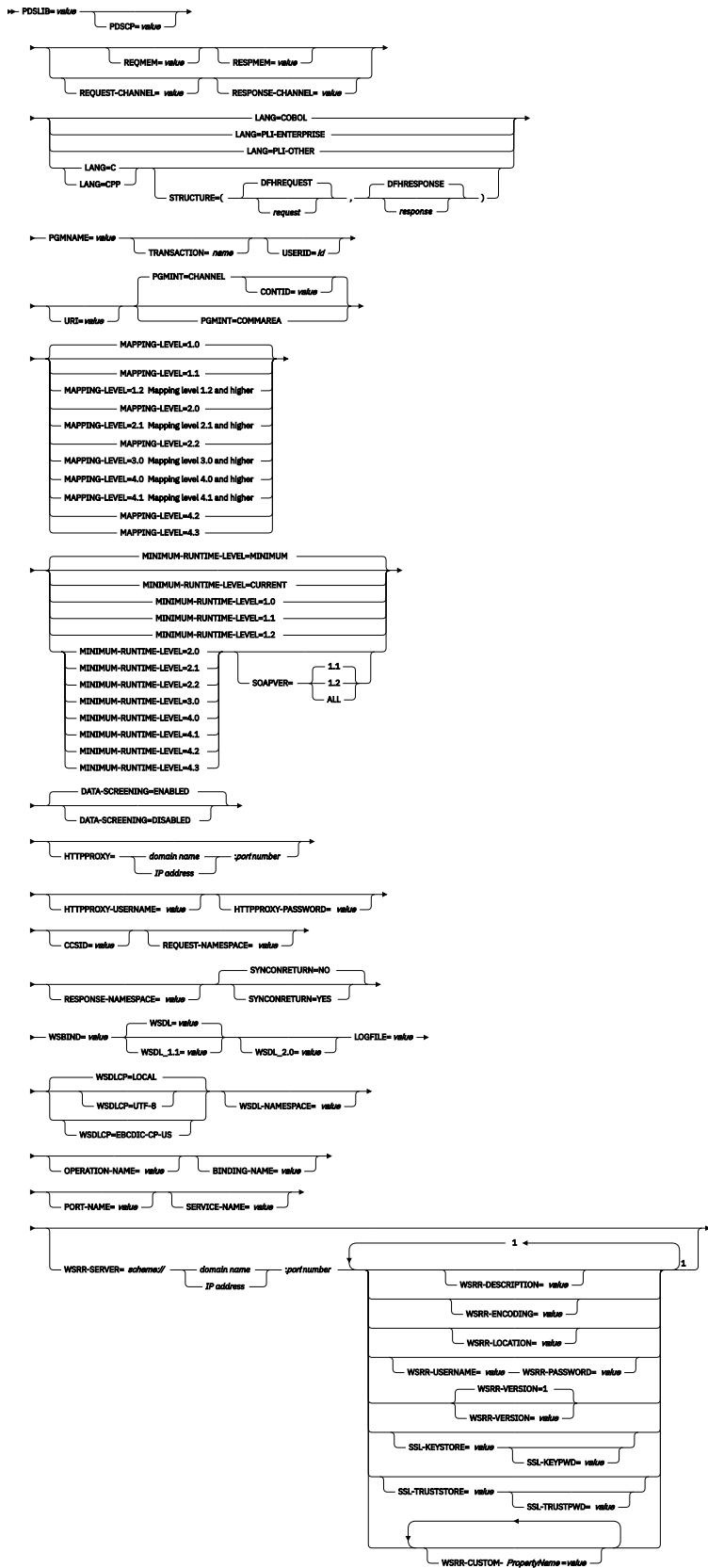
```
/tmp/LS2WS.in  
/tmp/LS2WS.out  
/tmp/LS2WS.err
```

**Important:** DFHLS2WS does not lock access to the z/OS UNIX files or the data set members. 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, which might cause unpredictable failures.

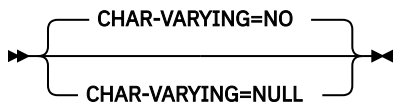
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of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

**Important:** The utility will only run under the default USS shell. If you have configured a different shell (such as the bash shell) to launch, based on your .profile, then the utility will fail to run without issuing any error messages. If you use a different shell you must alter the configuration of your .profile to use the default USS shell before running the utility.

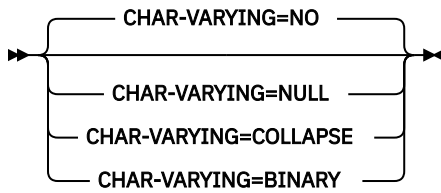
# Input parameters for DFHLS2WS



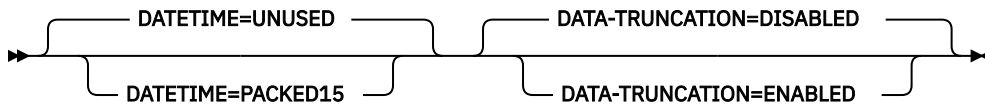
## Mapping level 1.2 and higher



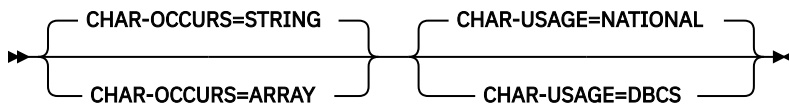
**Mapping level 2.1 and higher**



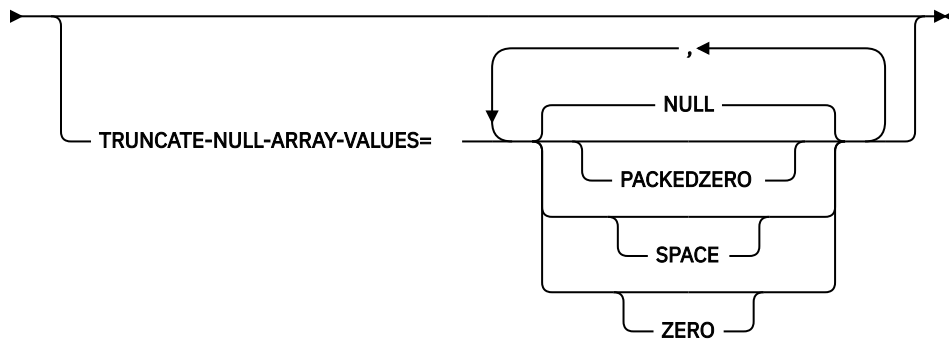
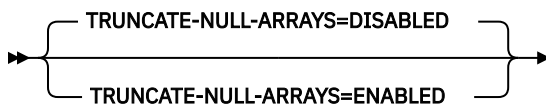
**Mapping level 3.0 and higher**



**Mapping level 4.0 and higher**



**Mapping level 4.1 and higher**



**Notes:**

<sup>1</sup> Each WSRR parameter that can be specified when the **WSRR-SERVER** parameter is set can be specified only once. The exception 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

### **BINDING-NAME = value**

Specifies the binding name that is used in the generated WSDL document. If no value is supplied, a default binding name is generated using the value of the **PGMNAME** parameter followed by "HTTPSoapBinding". If SOAPVER is set to ALL, a "12" suffix is appended to the name of the SOAP 1.2 binding.

### **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.

### **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 and embedded white space in the field is not included in the SOAP message; for example `<space>AB<space><space><space>C<space>` becomes `AB<space>C`. 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 behavior 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 DBCS encoded data.

### **NATIONAL**

Data from PIC ( n ) fields is treated as UTF-16 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-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

### **ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

### **DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

## **DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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 | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

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 or the web service once it is deployed.

**MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

Specifies the level of mapping that DFHLS2WS uses when generating the web service binding file and web service description. You can select these options. Each level of mapping inherits the function of the previous mapping, where the highest level of mapping offers the best capabilities available. See “Mapping levels for the CICS assistants” on page 332 for minimum CICS release level required by each mapping level and other compatibility information.

**1.0**

This mapping level is the default.

**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 to take advantage of the enhancements to the mapping between the language structure and web services binding file.

## 2.1

Use this mapping level to 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 to take advantage of mapping enhancements when using DFHWS2LS.

## 3.0

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 region at CICS TS 5.3 or later. At this mapping level you can use COBOL OCCURS DEPENDING ON fields and the **CHAR-OCCURS** parameter.

## 4.1

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

## 4.2

No significant changes. Use this mapping level with a region at CICS TS V5.4 or later.

## 4.3

No significant changes. Use this mapping level with a region at CICS TS V5.4 or later.

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 | 4.1 | 4.2 | 4.3 | 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 selected.

### 1.0

The generated web service binding file deploys successfully into a CICS TS for z/OS, 3.1 region. Some parameters are not available at this runtime level.

### 1.1

The generated web service binding file deploys successfully into a CICS TS for z/OS, 3.1 region. 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 for z/OS, 3.1 region. 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 for z/OS, 3.2 or later region. 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 region at CICS TS for z/OS, 3.2 or later. 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 region at CICS TS for z/OS, 3.2 region or later. 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 region at CICS TS for z/OS, Version 4.1, 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 region at CICS TS for z/OS, Version 5.3, 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.

### 4.1

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

### 4.2

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

### 4.3

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

### 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, a default name is generated using the value of the **PGMNAME** parameter followed by the **operation** value.

### **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.

**PORT-NAME = value**

Specifies the name that is used for the port and portType in the generated WSDL document. If no value is supplied, a default name is generated using the value of the **PGMNAME** parameter followed by "Port" . If SOAPVER is set to ALL, a "12" suffix is appended to the name of the SOAP 1.2 port.

**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.

**SERVICE-NAME = value**

Specifies the service name that is used in the generated WSDL document. If no value is supplied, a default service name is generated using the value of the **PGMNAME** parameter followed by "Service".

**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 \$ @ # \_ < >

**TRUNCATE-NULL-ARRAYS = { DISABLED | ENABLED }**

Specifies how structured arrays are processed at mapping level 4.1 or higher. If enabled, CICS will attempt to recognize empty records within an array (see TRUNCATE-NULL-ARRAY-VALUES for more

information about identifying empty records). If five consecutive empty array records are detected, the array is truncated at the first such record when generating XML/JSON. This truncation capability is only enabled for arrays with structured content, arrays of simple primitive fields are not subject to truncation. Truncation of arrays can result in a more concise representation of the data in JSON/XML, but is not without risk. If five consecutive data records are misidentified as uninitialized storage (perhaps because they legitimately contain low values), data loss can be experienced. If TRUNCATE-NULL-ARRAYS is enabled and TRUNCATE-NULL-ARRAY-VALUES is not set, then the default value for TRUNCATE-NULL-ARRAY-VALUES is used.

**TRUNCATE-NULL-ARRAY-VALUES = { NULL | PACKEDZERO | SPACE | ZERO }**

Specifies which values are treated as empty for TRUNCATE-NULL-ARRAYS processing at mapping level 4.1 or higher. By default, the null value (0x00, or low-values) is treated as empty. If all of the bytes of storage within a record of a structured array contain nulls, then the entire record is considered to be empty. One or more of the NULL, PACKEDZERO, SPACE and ZERO values can be specified in a comma separated list.

**NULL**

Implies a null character (0x00).

**PACKEDZERO**

Implies a positive signed packed decimal zero (0x0C), a negative signed packed decimal zero (0x0D), or an unsigned packed decimal zero (0x0F).

**SPACE**

Implies an SBCS EBCDIC space (0x40).

**ZERO**

Implies an unsigned zoned decimal zero (0xF0).

Any matching combination of the selected bytes within a structured array record will cause the entire record to be identified as empty.

If TRUNCATE-NULL-ARRAY-VALUES has a value defined, TRUNCATE-NULL-ARRAYS must be enabled.

**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 **WSRR-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's 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/wsbinding/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/wsbinding/example.wsbinding
WSDL=/u/exampleapp/wsd1/example.wsd1
WSDL_2.0=/u/exampleapp/wsd1/example_20.wsd1
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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

**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 **PATHPREF** 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`.

### **PATHMAIN = *path***

Specifies the main part of the name of the CICS TS directory in the UNIX system services file system.

The default value is `/usr/lpp/cicsts`.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHMAIN** parameter.

### **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 the value specified by the **PATHMAIN** parameter.

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
```

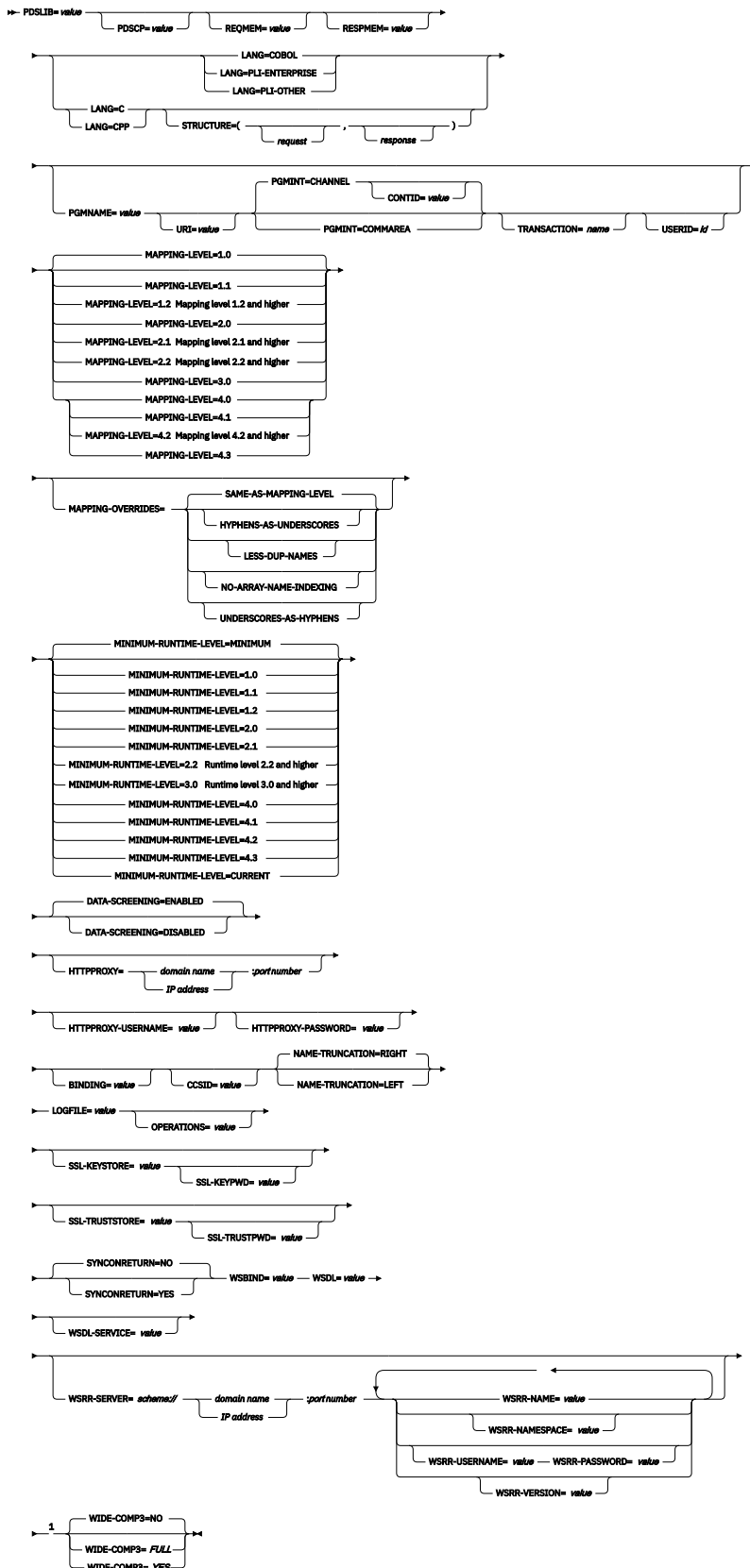
**Important:** DFHWS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHWS2LS run concurrently, and use the same temporary workspace files,

nothing prevents one job overwriting the workspace files while another job is using them, leading to unpredictable failures.

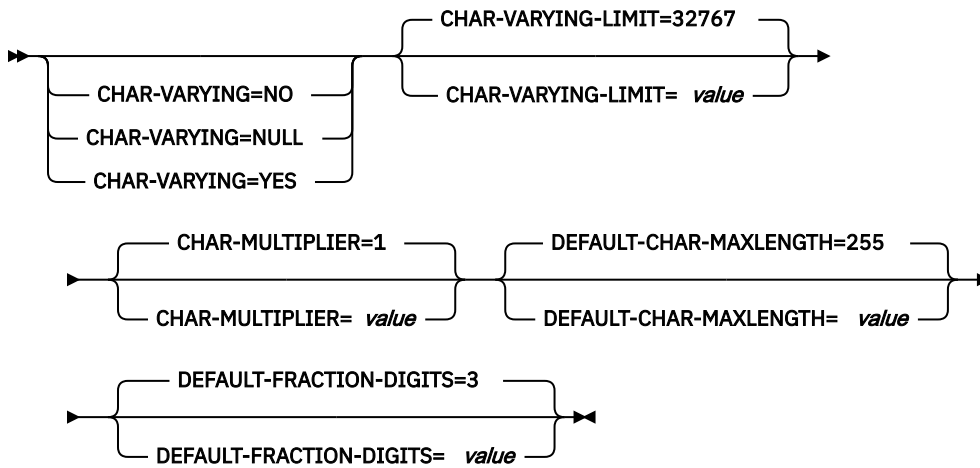
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of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

**Important:** The utility will only run under the default USS shell. If you have configured a different shell (such as the bash shell) to launch, based on your .profile, then the utility will fail to run without issuing any error messages. If you use a different shell you must alter the configuration of your .profile to use the default USS shell before running the utility.

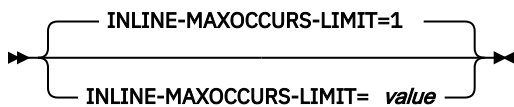
# Input parameters for DFHWS2LS



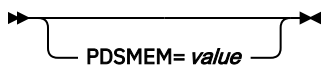
## Mapping level 1.2 and higher



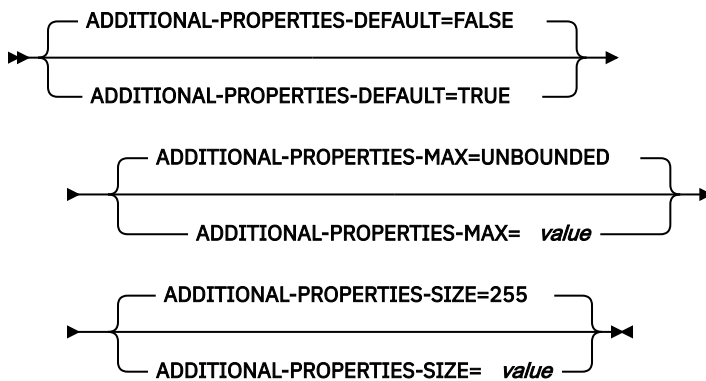
### Mapping level 2.1 and higher



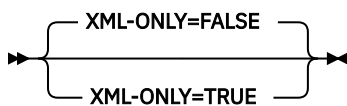
### Mapping level 2.2 and higher



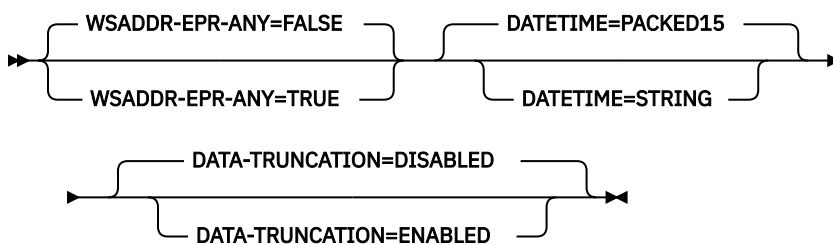
### Mapping level 4.2 and higher



### Runtime level 2.1 and higher



### Runtime level 3.0 and higher



Notes:

<sup>1</sup> 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

### **ADDITIONAL-PROPERTIES-DEFAULT = { true | false }**

Indicates whether JSON schema objects that do not explicitly declare support for Additional Properties are interpreted as supporting them or not. Additional JSON properties are any properties within a JSON object that are not pre-defined in the JSON Schema. These properties are typically rejected by the data transformation mechanism as unexpected extra data. If **ADDITIONAL-PROPERTIES-DEFAULT** is set to TRUE, or if the JSON schema explicitly sets `additionalProperties:true` for an object, then space is allocated in the generated copybooks to hold such values. Applications can interact with those values using the associated fields in the copybooks.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-MAX = { 0-20 | UNBOUNDED }**

Indicates how many Additional Properties are supported for a JSON object that supports them. See **ADDITIONAL-PROPERTIES-DEFAULT**. The generated copybooks will contain structures suitable for addressing any additional properties. By default, there is no maximum constraint placed on the number of properties that are supported. The copybooks are generated in a similar fashion to arrays with no constraints and use containers. This parameter can be used to apply a maximum constraint that can be used in combination with the **INLINE-MAXOCCURS-LIMIT** parameter to cause a fixed length array to be allocated for the maximum number of properties, thereby avoiding the need for containers.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-SIZE = { 16-32767 | 255 }**

Indicates the maximum size for each of the JSON additional properties. If a JSON object supports additional properties, as defined by **ADDITIONAL-PROPERTIES-DEFAULT**, then the generated copybooks will have bindings to support properties up to the number specified by **ADDITIONAL-PROPERTIES-MAX**. By default, the maximum value supported for each additional property is 255 characters. A field of that size will be generated into the copybooks that are produced. This size can be customized by setting the **ADDITIONAL-PROPERTIES-SIZE** parameter. For example, a JSON object is processed that is found to contain the following property:

```
"example": { "notes": "this extra property was not defined in the JSON Schema" }
```

If the copybooks have been generated to support additional properties, then that entire value will be passed to the application for processing. The value begins with the leading quotation mark before the property's key, and ends with the trailing right brace in the property's value. It's approximately

100 characters in this example. The value used for **ADDITIONAL - PROPERTIES - SIZE** must be large enough to hold the largest such value that might occur. If the allocated buffer is too small for the value that is processed, an error response is generated.

You can use this parameter at a mapping level of 4.2 or higher.

**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.

**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.

**Note:** If PGMINT is set to COMMAREA, variable length strings cannot be mapped into containers, and this parameter has no effect.

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-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

**ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

**DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

**DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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 | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

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 or the web service once it is deployed.

**MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

Specifies the level of mapping that DFHWS2LS uses when generating the web service binding file and language structure. You can select these options. Each level of mapping inherits the function of the previous mapping, where the highest level of mapping offers the best capabilities available. See [“Mapping levels for the CICS assistants” on page 332](#) for minimum CICS release level required by each mapping level and other compatibility information.

**1.0**

This mapping level is the default.

## 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 to take advantage of the enhancements to the mapping between the language structure and web services binding file.

## 2.1

Use this mapping level 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 to take advantage of the following support:

- Elements with fixed values
- Enhanced support for `<xsd:choice>` elements
- Abstract data types
- Abstract elements
- Substitution groups

## 3.0

At this mapping level you can transform timestamps to CICS ABSTIME format.

## 4.0

Use this mapping level with a region at CICS TS 5.3 or later when you want to use UTF-16.

## 4.1

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

## 4.2

Use this mapping level with a region at CICS TS V5.4 or later.

## 4.3

Use this mapping level with a region at CICS TS V5.4 or later.

For more information about mapping levels, see [Mapping levels for the CICS assistants](#).

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

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.

### **SAME-AS-MAPPING-LEVEL**

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

### **HYPHENS-AS-UNDERSCORES**

For PL/I only. This parameter converts any hyphens in the WSDL document to underscores rather than the character X, to improve the readability of the generated PL/I language structures. For more information, see [XML schema to PL/I mapping](#). This option is enabled automatically at mapping level 4.2.

## 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 PL/I 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),
```

This option is enabled automatically at mapping level 4.2.

## NO-ARRAY-NAME-INDEXING

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

## UNDERSCORES-AS-HYPHENS

This option is enabled automatically at mapping level 4.0.

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](#).

## MINIMUM-RUNTIME-LEVEL = { **MINIMUM** | **1.0** | **1.1** | **1.2** | **2.0** | **2.1** | **2.2** | **3.0** | **4.0** | **4.1** | **4.2** | **4.3** | **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 selected.

### 1.0

The generated web service binding file deploys successfully into a CICS TS for z/OS, 3.1 region. Some parameters are not available at this runtime level.

### 1.1

The generated web service binding file deploys successfully into a CICS TS for z/OS, 3.1 region. 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 for z/OS, 3.1 region. 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 for z/OS, 3.2 or later region. 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 region at CICS TS for z/OS, 3.2 or later. 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 region at CICS TS for z/OS, 3.2 region or later. 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 region at CICS TS for z/OS, Version 4.1, 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 region at CICS TS for z/OS, Version 5.3, 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.

## 4.1

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

## 4.2

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

## 4.3

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

## 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. The member name is generated by appending a number to the prefix, with a maximum member name length of 8 characters.

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. The member name is generated by appending a number to the prefix, with a maximum member name length of 8 characters.

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. The member name is generated by appending a number to the prefix, with a maximum member name length of 8 characters.

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.

**Example: Use of this input parameter for DFHWS2LS**

```
SSL-KEYSTORE=SAFKEYRING://USERID/Ring.name
```

Java 11 Use `safKeyringjce` when running Java 11:

```
SSL-KEYSTORE=SAFKEYRINGJCE://USERID/Ring.name
```

6.3 **Beta** Support for Java 8 and Java 11 is removed as of CICS TS 6.3.

**SSL-KEYPWD = value**

This optional parameter specifies the password for the Java key store. You do not need to use this when using a SAF keyring.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network.

**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.

**Example: Use of this input parameter for DFHWS2LS**

```
SSL-TRUSTSTORE=SAFKEYRING://USERID/Ring.name
```

Java 11 Use `safKeyringjce` when running Java 11:

```
SSL-TRUSTSTORE=SAFKEYRINGJCE://USERID/Ring.name
```

6.3 **Beta** Support for Java 8 and Java 11 is removed as of CICS TS 6.3.

**SSL-TRUSTPWD = value**

This optional parameter specifies the password for the Java trust store. You do not need to use this when using a SAF keyring.

Use this parameter if you want the web services assistant to use secure sockets layer (SSL) encryption to communicate across a network.

**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 *request nn* and *response nn* 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 | FULL | YES }**

Controls the maximum size of the packed decimal variable length in the generated COBOL or PL/I language structure.

**FULL**

For COBOL and PL/I. DFHJS2LS generates a packed decimal field that is large enough to hold all valid values. The maximum size is 31 digits. This is the default.

**NO**

For COBOL only. 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**

For COBOL only. DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

**Note:** The NO and YES options generate fields that are unable to represent all valid values; the FULL option avoids this problem. However, the FULL option does allow some invalid values to be represented in the packed decimal field. For example, if a schema indicates that there are a maximum of five digits and a maximum of two fractional digits, the FULL option will generate a packed decimal field that allows for seven digits, and this allows space for valid values such as 25000 and 999.99, but also provides space for some invalid values such as 9999.99. When you use the FULL option, take care not to generate invalid values in application data.

**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. The file name is restricted to characters that are valid for a URL and in particular may not contain any `#` character. 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/wsd1/example.wsd1
/*
```

### Parameters required by DFHWS2LS to support WS-Addressing

When you configure your WSDL for Web Services Addressing, you must set the **MINIMUM-RUNTIME** and the **MAPPING-LEVEL** parameters on the web services assistant, DFHWS2LS, to a value of 3.0 or higher. You might also want to consider setting the **WSADDR-EPR-ANY** parameter to TRUE.

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 WsBind files that the assistant generates fully support web services addressing and can interoperate with other web services platforms.

Set the **MAPPING-LEVEL** parameter on the web services assistant, DFHWS2LS, to 3.0 or higher.

If you have any elements of type `wsa:EndpointReferenceType` in the request or response messages defined in your WSDL document, and you want to use these elements as input to the **WSACONTEXT BUILD** API command at run time, set the **WSADDR-EPR-ANY** parameter to TRUE. 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 must treat the EPR data as an `<xsd:any>` element and store 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 will be stored in a named container as an `<xsd:any>` element at runtime 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.

## 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

### 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 154.](#)

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.](#)

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, see [Working 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](#).

## 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 176.
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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

### 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 IBM 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 141](#).

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](#).
  - 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 177](#).
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](#).

## 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" 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.

## Procedure

Follow these steps to customize generated web service description documents:

1. If you want to advertise support for HTTPS or communicate using IBM 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 TCPIP SERVICE resource.
  - b) If you are using IBM 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.

## 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.
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 &lt; . 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 FAULTSTRING 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.

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

## 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

### 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 154.
- 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](#).

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 IBM 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](#).

## Results

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

## Developing web services using tooling

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

### Procedure

1. You have two choices:

- Use the IBM Developer for z/OS tool to create a web service binding file and the web service description or language structures. For more information about this tool, see [IBM Developer for z/OS](#).
- Write your own Java program, using the provided API, to invoke the web services assistant. This API is described in [Web services assistant Javadoc reference](#). 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 177.

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.

### Web services assistant Javadoc reference

[6.2 Web services assistant Javadoc reference](#)

[6.1 Web services assistant Javadoc reference](#)

## 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 IBM Developer for z/OS 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 web services pipelines](#).

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.

## Testing CICS SOAP Web services

You can quickly and easily test a Web service without writing a complete client side application just to confirm that the CICS configuration is correct.

### Before you begin

Before you can begin to test your Web service:

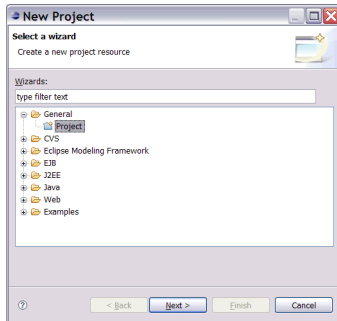
- You must have created a WSDL document using DFHLS2WS that describes a Web service hosted in CICS.
- You must have deployed a WSBind file into CICS.

## About this task

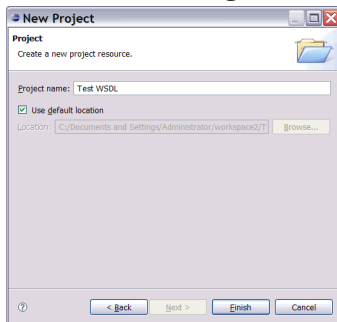
Many tools and products exist for working with WSDL, some of which are free. What follows is a tutorial on how to use the Eclipse development environment to test Web services. In this example, Eclipse V3.3 and the optional Eclipse Web tools plug-ins are being used.

## Procedure

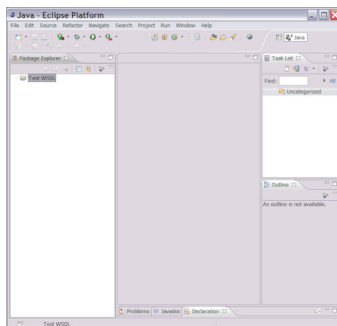
1. Start a new project by choosing **File > New > Project** and select a **General** project.



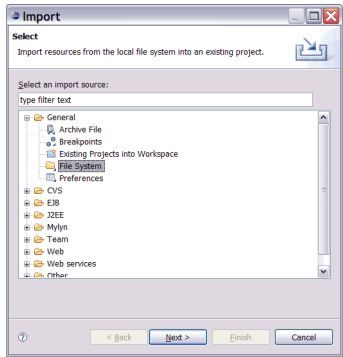
2. Click on **Next** and give it a name. In this example, the project is called 'Test WSDL'.



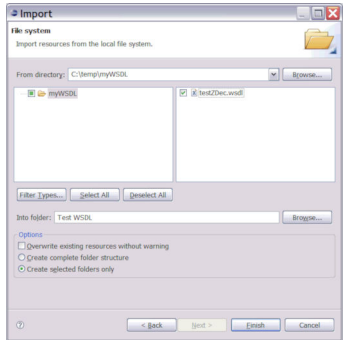
3. Click **Finish**. You will see that an empty project has been created in the **Package Explorer**.



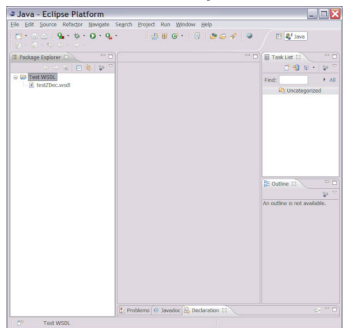
4. Right-click on the project and select **Import**. Then select a **General** import from the **File System**.



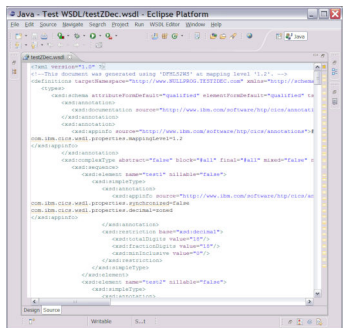
5. Navigate to the WSDL document you wish to use:



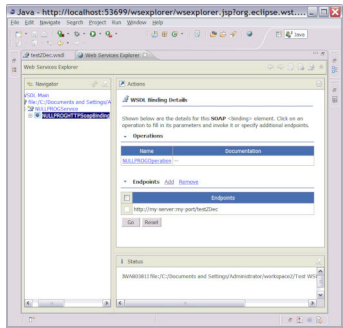
6. Click **Finish** and you will see your WSDL document in the **Package Explorer**.



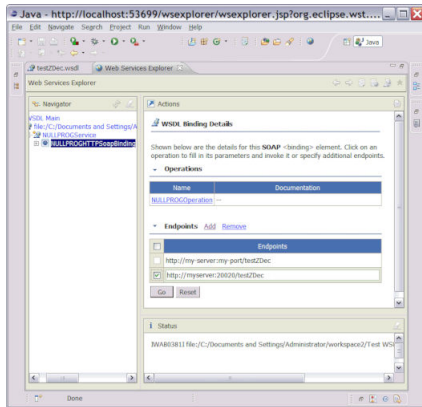
7. You can now view the WSDL in Eclipse:



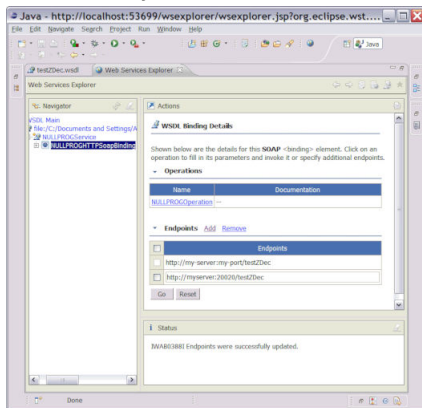
8. Right-click on your WSDL file in Eclipse - in this case it is `testZDec.wsdl` - and click **Web Services > Test with Web Services Explorer**. Eclipse generates a web-based client for the remote Web service.



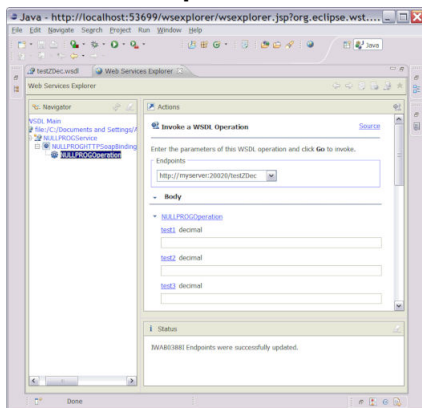
9. Before you can call the Web service you must add the real endpoint information (something that you could have put in the WSDL before importing it). Do this by clicking **Add** and entering the URL that CICS is listening on:



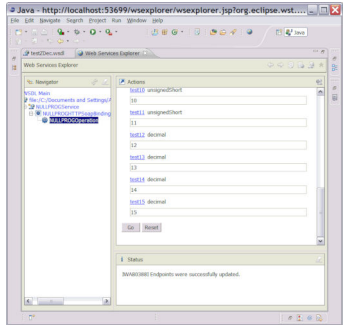
10. Click **Go** and message IWAB0388I appears in the **Status** area, announcing that the endpoint was successfully added:



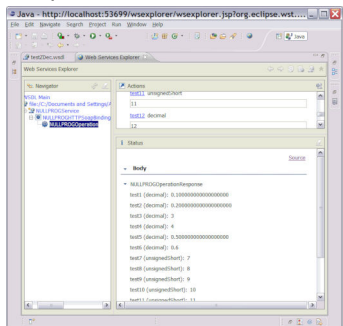
11. Use the Navigator to select the operation you wish to invoke. In this example, there is only one option - **NULLPROGOperation**.



12. Now you can supply values for all of the input parameters. In this example, the parameters are called **test1**, **test2** and so on.

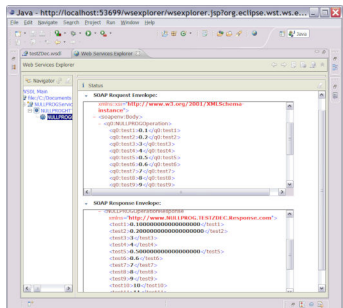


13. Now click **Go** to send the SOAP message to CICS. The output from the Web service is displayed in Eclipse:

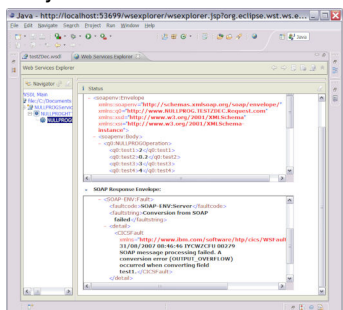


In this example, the NULLPROG program in CICS returns the same data it was invoked with.

14. Click on the **source** link to view the actual SOAP messages that flowed on the wire:



15. In this example, you can see that a SOAP Fault message was returned from CICS (field **test1** can only have values between 0 and 1, so the value 2 is too large):



## Results

This quick demonstration has shown what can be done using free tools to test Web services hosted in CICS.

## 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. Set up an OSGi JVM server in the CICS region. SOAP validation using DFHPIVAL will only run in an OSGi framework, not in an Axis2 or Liberty profile JVM.
  - a) Install the sample JVM server DFH\$JVMS in group DFH\$OSGI or create your own JVM server.  
For more information, see [Setting up a JVM server](#).
  - b) If you created your own JVM server, 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 DFH\$JVMS.
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.

## 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.

**Important:** Axis2 Java provider-mode web services are only supported with Axis2 JVM servers running Java 8.

**Note:** Axis2-style applications do not use the WEBSERVICE resources. They interact with CICS using the Axis2 programming model and therefore cannot use 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](#)
- [DFHWS-MEP container](#)
- [DFHWS-USERID container](#)
- [DFHWS-TRANID container](#)
- [Security for SOAP web services](#)

## 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
 */
@XmlWebService(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](#).
  - 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
SStatus : 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 185](#) or [“Creating an XML-aware service provider application” on page 184](#).

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](#).
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](#).

### 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 IBM Z 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. This topic applies to CICS integrated-mode Liberty only.

### About this task

CICS TS V5.3 includes the latest WebSphere Liberty 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 more information, see the CICS DevCenter article, [JAX-WS web service sample for Liberty](#).

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 IBM Z Application Assist Processor (zAAP).

**Note:** Web applications deployed to a Liberty JVM server do not use the WEBSERVICE or TCPIP SERVICE resources. They interact with web requests using the Liberty HTTP listener and therefore cannot use the facilities of the CICS web services support.



---

## Chapter 4. Developing with JSON

For any purpose, you can write application programs to transform application binary data into JavaScript Object Notation (JSON) and vice versa. CICS supports a number of high-level languages and provides a JSON assistant to map how the data is transformed during runtime processing. CICS uses the same technology for mapping application data to JSON messages, as part of the web services support.

### Before you begin

You must have Java installed to run the JSON assistant. Transformations can be carried out either internally with CICS or by using a JVM server. If you use Java for the transformations, then you must have an Axis2 JVM server installed to transform application data and JSON. See [Configuring a JVM server for Axis2](#) for more information.

### About this task

The advantage of using this approach to transform application data to and from JSON is that CICS goes beyond the capabilities that are offered by a JSON parser. CICS can interpret the JSON and perform record-based conversions of the application data. Therefore, it is easier and faster for you to create applications that work with JSON by using this approach.

The CICS JSON assistant is a supplied utility that helps you to create the required mapping artifacts to transform application binary data to JSON or to transform JSON to application binary data. The JSON assistant creates the artifacts in a bundle directory.

### Procedure

1. Create a bundle using the JSON assistant.  
This bundle contains the required mapping artifacts for data transformation.
2. Install the bundle in CICS to make the mappings available.
3. Create or update an application program to handle data transformation.  
You have two options:
  - Use the **TRANSFORM DATATOJSON** and **TRANSFORM JSONTODATA** API commands in the application program. This is the recommended approach.
  - Use the **LINK PROGRAM** API command to link to CICS -supplied linkable interface DFHJSON.

The application must use a channel-based interface.

4. Run the application to test that the transformation works as you intended.

Steps “1” on page 197 to “4” on page 197 are explained in more detail in the following topics.

---

## The CICS JSON assistant

The CICS JSON assistant is a set of batch utilities that generate mappings between high-level language structures and JSON schemas for performing transformations between JSON and application data. The assistant supports rapid deployment of applications that perform JSON processing, for use in service providers and service requesters, with minimum amount of programming effort.

When you use the CICS 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 JSON assistant can create a JSON schema from a simple language structure, or a language structure from an existing XML schema, and supports COBOL, C/C++, and PL/I. It also generates metadata that CICS uses at run time to automatically convert XML data to binary application data or vice versa; the

metadata is defined in an XML binding and stored on z/OS UNIX. The schema for the XML binding is in /usr/lpp/cicsts/cicsts52/schemas/xmltransform/ directory on z/OS UNIX.

The CICS JSON assistant comprises two utility programs:

#### **DFHLS2JS**

DFHLS2JS maps high-level language structures to JSON schemas. It also generates a web service binding file from a language structure.

#### **DFHJS2LS**

DFHJS2LS maps JSON schemas to high-level language structures. This utility generates a web service binding file from a JSON schema and a language structure that you can use in your application programs.

The JCL procedures to run both programs are in the *hlq*.XDFHINST library, where *hlq* is the high-level qualifier of your CICS installation.

You must code high-level language structures that are processed by DFHLS2JS according to the rules of the language, as implemented in the language compilers that CICS supports.

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](#)

### **Data mappings**

The mapping from high-level language structures to JSON schemas and the mapping from JSON schemas to high-level language structures are not symmetrical.

For detailed information, see the following topics:

- [“How the CICS JSON assistant maps between high-level languages and JSON schemas” on page 253](#)
- [“Mapping levels for the CICS JSON assistant ” on page 253](#)
- [“Variable arrays of elements in DFHJS2LS” on page 296](#)

## **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 DFHLS2JS JCL procedure is installed in the data set *HLQ*.XDFHINST, where *HLQ* is the high-level qualifier where CICS is installed.

### **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`.

### **PATHMAIN = *path***

Specifies the name of the CICS TS directory in the UNIX system services file system. The value of this parameter is appended to the value specified by the **USSDIR** parameter.

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.

### **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 the value specified by the **PATHMAIN** parameter. 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
```

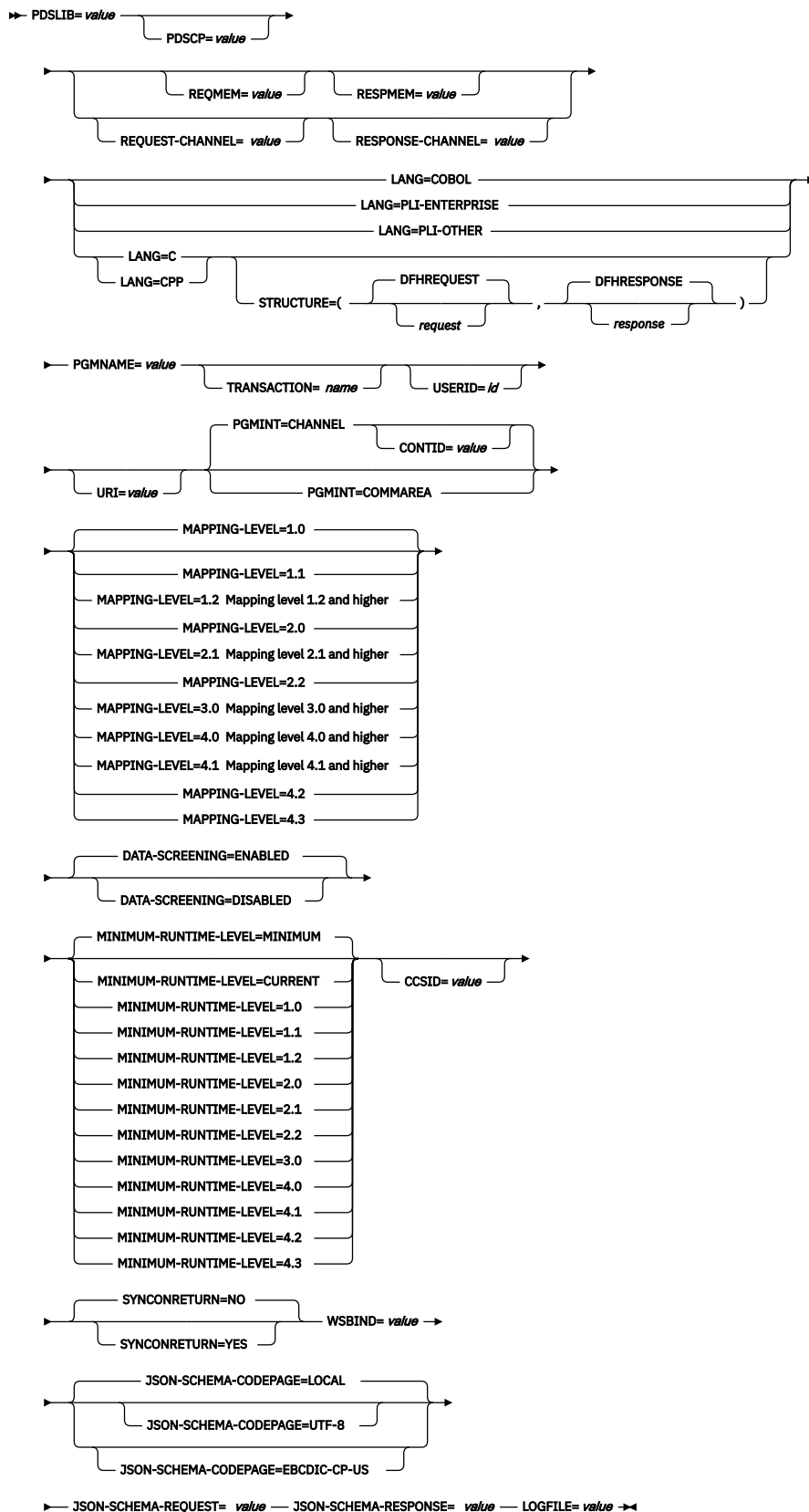
**Important:** DFHLS2JS does not lock access to the z/OS UNIX files or the data set members. If two or more instances of DFHLS2JS run concurrently and use the same temporary workspace files, nothing

prevents one job from overwriting the workspace files while another job is using them, which might cause unpredictable failures.

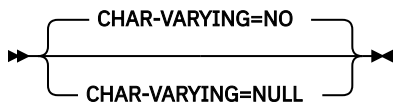
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of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

**Important:** The utility will only run under the default USS shell. If you have configured a different shell (such as the bash shell) to launch, based on your .profile, then the utility will fail to run without issuing any error messages. If you use a different shell you must alter the configuration of your .profile to use the default USS shell before running the utility.

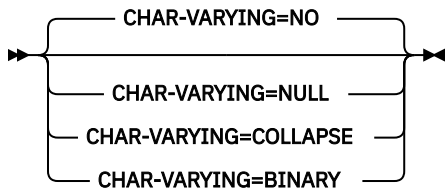
# Input parameters for DFHLS2JS



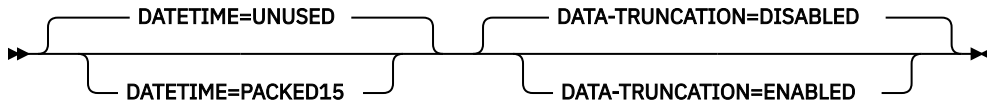
## Mapping level 1.2 and higher



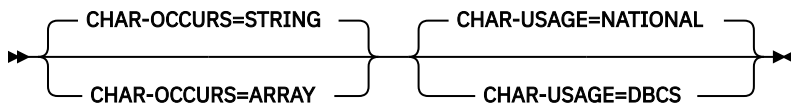
**Mapping level 2.1 and higher**



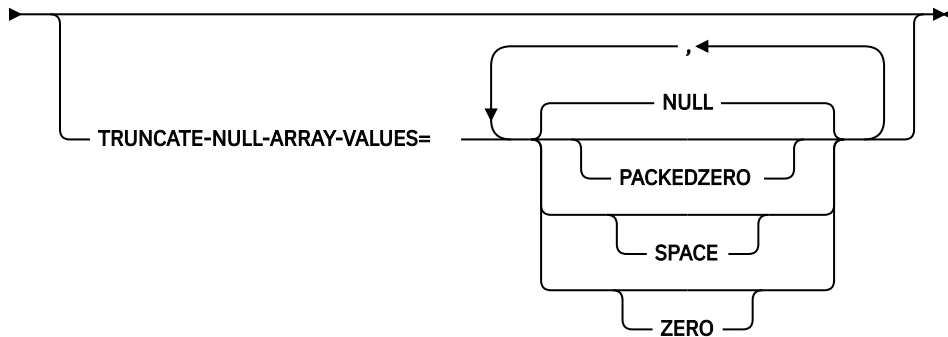
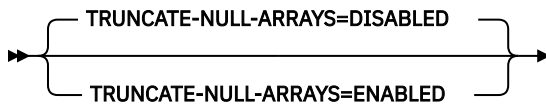
**Mapping level 3.0 and higher**



**Mapping level 4.0 and higher**



**Mapping level 4.1 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-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 and embedded white space in the field is not included in the JSON message; for example <space>AB<space><space><space>C<space> becomes AB<space>C. 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.

For more information about handling variable-length values and white space, see [“Support for variable-length values and white space” on page 372](#).

### **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 a 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 DBCS encoded data.

**NATIONAL**

Data from PIC ( n ) fields is treated as UTF-16 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-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

**ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

**DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

**DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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-CODEPAGE = { LOCAL | UTF-8 | EBCDIC-CP-US }**

Specifies the code page that is used to generate the JSON Schema documents.

**LOCAL**

Specifies that the JSON Schemas are generated using the default code page for the file system.

**UTF-8**

Specifies that the JSON Schemas are generated using the UTF-8 code page.

**EBCDIC-CP-US**

Specifies that the JSON Schemas are generated using the US EBCDIC code page.

**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 | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

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 or the web service once it is deployed.

**MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

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 region at CICS TS 5.3 or later. At this mapping level you can use COBOL OCCURS DEPENDING ON fields and the **CHAR-OCCURS** parameter.

#### 4.1

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

#### 4.2

No significant changes. Use this mapping level with a region at CICS TS V5.4 or later.

#### 4.3

No significant changes. Use this mapping level with a region at CICS TS V5.4 or later.

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 | 4.1 | 4.2 | 4.3 | 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 region.

#### **4.0**

The generated web service binding file deploys successfully into a region at CICS TS 5.3 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.

#### **4.1**

The generated web service binding file deploys successfully into a region at CICS TS 5.3 or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

#### **4.2**

The generated web service binding file deploys successfully into a region at CICS TS 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

#### **4.3**

The generated web service binding file deploys successfully into a region at CICS TS 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

## **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 134.

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 134](#).

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 \$ @ # \_ < >

**TRUNCATE-NULL-ARRAYS = { DISABLED | ENABLED }**

Specifies how structured arrays are processed at mapping level 4.1 or higher. If enabled, CICS will attempt to recognize empty records within an array (see TRUNCATE-NULL-ARRAY-VALUES for more information about identifying empty records). If five consecutive empty array records are detected, the array is truncated at the first such record when generating XML/JSON. This truncation capability is only enabled for arrays with structured content, arrays of simple primitive fields are not subject to truncation. Truncation of arrays can result in a more concise representation of the data in JSON/XML, but is not without risk. If five consecutive data records are misidentified as uninitialized storage (perhaps because they legitimately contain low values), data loss can be experienced. If TRUNCATE-NULL-ARRAYS is enabled and TRUNCATE-NULL-ARRAY-VALUES is not set, then the default value for TRUNCATE-NULL-ARRAY-VALUES is used.

**TRUNCATE-NULL-ARRAY-VALUES = { NULL | PACKEDZERO | SPACE | ZERO }**

Specifies which values are treated as empty for TRUNCATE-NULL-ARRAYS processing at mapping level 4.1 or higher. By default, the null value (0x00, or low-values) is treated as empty. If all of the bytes of storage within a record of a structured array contain nulls, then the entire record is considered to be empty. One or more of the NULL, PACKEDZERO, SPACE and ZERO values can be specified in a comma separated list.

**NULL**

Implies a null character (0x00).

**PACKEDZERO**

Implies a positive signed packed decimal zero (0x0C), a negative signed packed decimal zero (0x0D), or an unsigned packed decimal zero (0x0F).

**SPACE**

Implies an SBCS EBCDIC space (0x40).

**ZERO**

Implies an unsigned zoned decimal zero (0xF0).

Any matching combination of the selected bytes within a structured array record will cause the entire record to be identified as empty.

If TRUNCATE-NULL-ARRAY-VALUES has a value defined, TRUNCATE-NULL-ARRAYS must be enabled.

**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's Guide](#).

**Example**

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
// SET QT=' '
//JAVAPROG EXEC DFHLS2JS,
// TMPFILE=&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. This topic lists the job control statements, symbolic parameters, input parameters, and their descriptions for DFHJS2LS.

The DFHJS2LS JCL procedure is installed in the data set *HLQ* .XDFHINST , where *HLQ* is the high-level qualifier where CICS is installed.

### 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*.

#### **PATHMAIN = *path***

Specifies the main part of the name of the CICS TS directory in the UNIX system services file system.

The default value is */usr/lpp/cicsts*.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHMAIN** parameter.

#### **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 the value specified by the **PATHMAIN** parameter. 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

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:

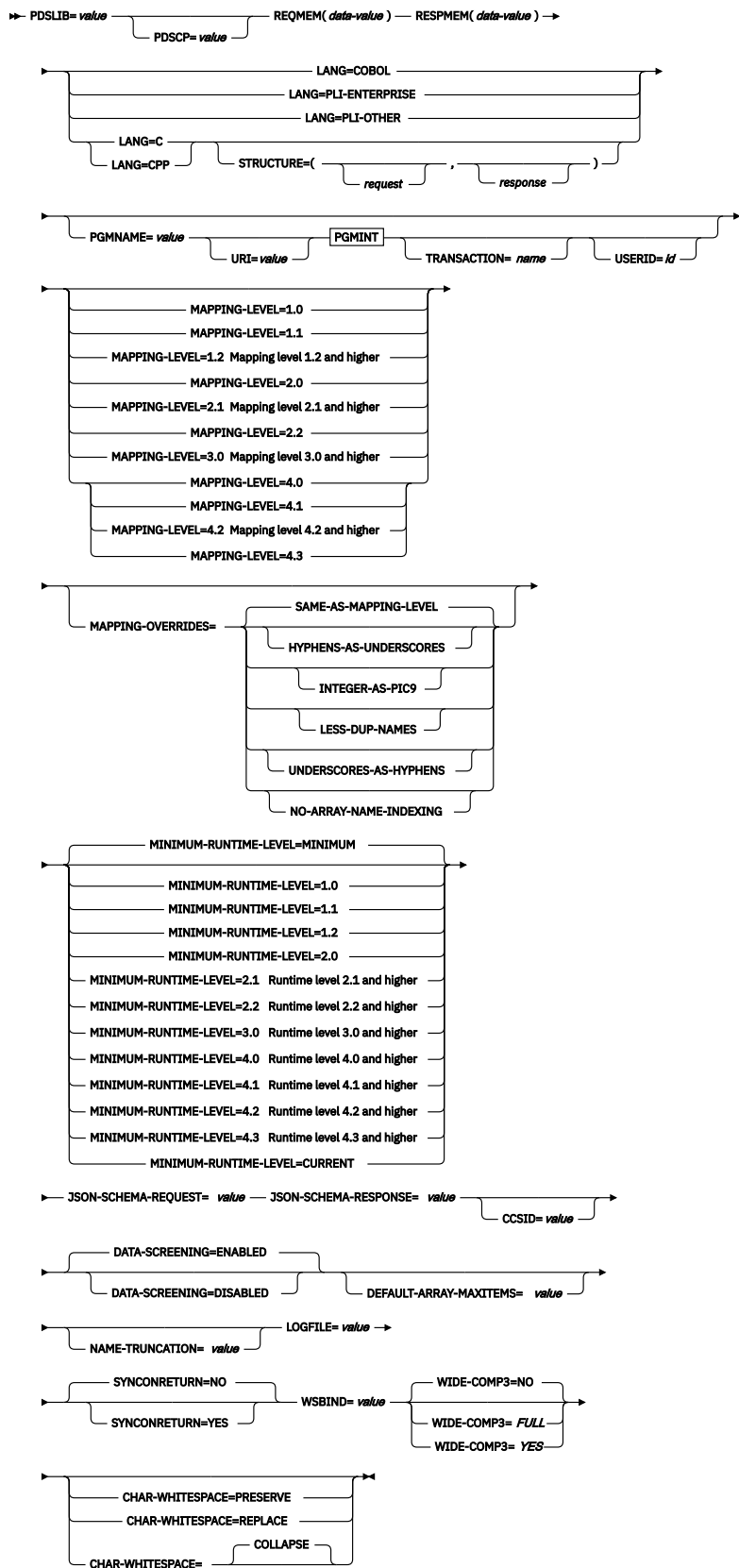
```
/tmp/JS2LS.in  
/tmp/JS2LS.out  
/tmp/JS2LS.err
```

**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.

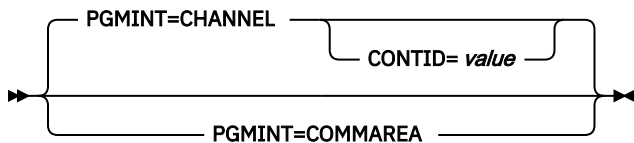
Therefore, you are advised to devise a naming convention, and operating procedures, that avoid this situation. For example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user. These temporary files are deleted before the end of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

**Important:** The utility will only run under the default USS shell. If you have configured a different shell (such as the bash shell) to launch, based on your .profile, then the utility will fail to run without issuing any error messages. If you use a different shell you must alter the configuration of your .profile to use the default USS shell before running the utility.

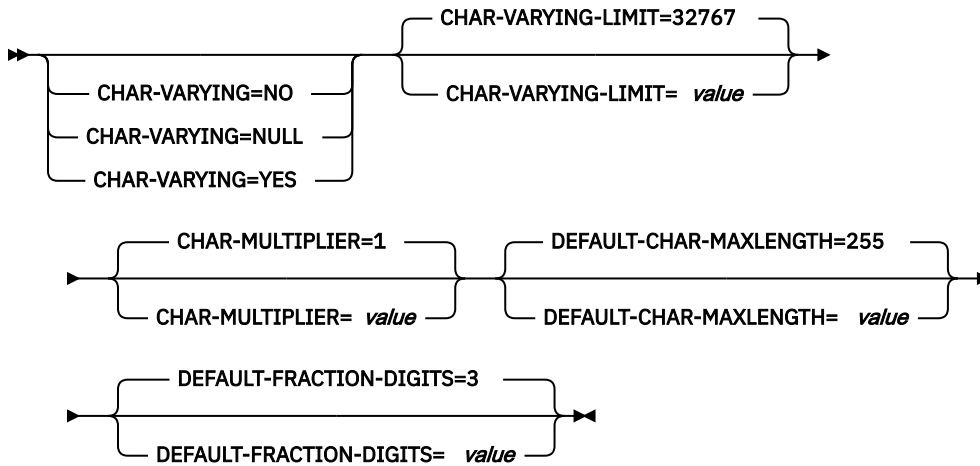
# Input parameters for DFHJS2LS



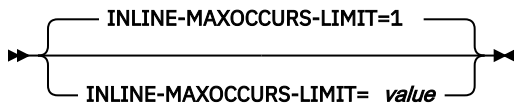
## PGMINT



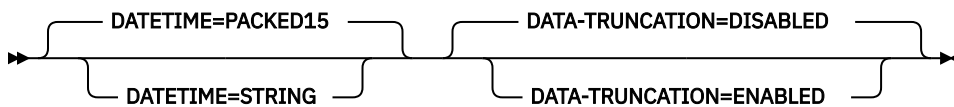
### Mapping level 1.2 and higher



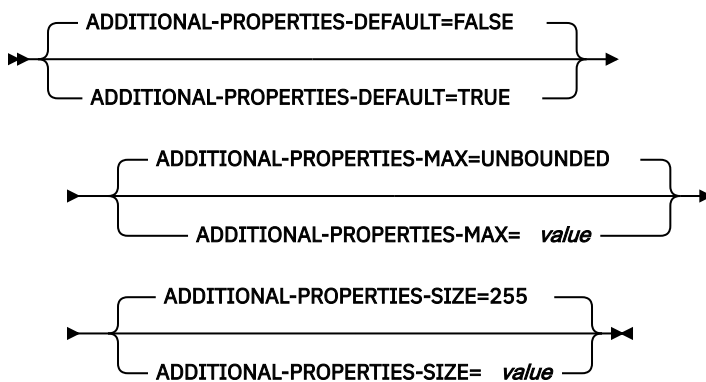
### Mapping level 2.1 and higher



### Mapping level 3.0 and higher



### Mapping level 4.2 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

### **ADDITIONAL-PROPERTIES-DEFAULT = { true | false }**

Indicates whether JSON schema objects that do not explicitly declare support for Additional Properties are interpreted as supporting them or not. Additional JSON properties are any properties within a JSON object that are not pre-defined in the JSON Schema. These properties are typically rejected by the data transformation mechanism as unexpected extra data. If **ADDITIONAL-PROPERTIES-DEFAULT** is set to TRUE, or if the JSON schema explicitly sets `additionalProperties:true` for an object, then space is allocated in the generated copybooks to hold such values. Applications can interact with those values using the associated fields in the copybooks.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-MAX = { 0-20 | UNBOUNDED }**

Indicates how many Additional Properties are supported for a JSON object that supports them. See **ADDITIONAL-PROPERTIES-DEFAULT**. The generated copybooks will contain structures suitable for addressing any additional properties. By default, there is no maximum constraint placed on the number of properties that are supported. The copybooks are generated in a similar fashion to arrays with no constraints and use containers. This parameter can be used to apply a maximum constraint that can be used in combination with the **INLINE-MAXOCCURS-LIMIT** parameter to cause a fixed length array to be allocated for the maximum number of properties, thereby avoiding the need for containers.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-SIZE = { 16-32767 | 255 }**

Indicates the maximum size for each of the JSON additional properties. If a JSON object supports additional properties, as defined by **ADDITIONAL-PROPERTIES-DEFAULT**, then the generated copybooks will have bindings to support properties up to the number specified by **ADDITIONAL-PROPERTIES-MAX**. By default, the maximum value supported for each additional property is 255 characters. A field of that size will be generated into the copybooks that are produced. This size can be customized by setting the **ADDITIONAL-PROPERTIES-SIZE** parameter. For example, a JSON object is processed that is found to contain the following property:

```
"example": { "notes": "this extra property was not defined in the JSON Schema" }
```

If the copybooks have been generated to support additional properties, then that entire value will be passed to the application for processing. The value begins with the leading quotation mark before the property's key, and ends with the trailing right brace in the property's value. It's approximately 100 characters in this example. The value used for **ADDITIONAL-PROPERTIES-SIZE** must be large enough to hold the largest such value that might occur. If the allocated buffer is too small for the value that is processed, an error response is generated.

You can use this parameter at a mapping level of 4.2 or higher.

### **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 non-numeric 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.

**CHAR-WHITESPACE = COLLAPSE | REPLACE | PRESERVE**

Specifies how white space in values of type string is handled by CICS.

**COLLAPSE**

Leading, trailing, and embedded white space is removed and all tabs, new lines, and consecutive spaces are replaced with single space characters.

**REPLACE**

Any tabs or new lines are replaced with the appropriate number of spaces.

**PRESERVE**

Retains any white space in the data value.

If the **CHAR-WHITESPACE** parameter is not set, white space is collapsed.

**Note:** This parameter does not apply to fields with a format of date-time, uri, base64Binary, or hexBinary, where white space is always collapsed.

**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-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

**ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

**DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

**DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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 time stamp and is mapped to CICS ABSTIME format.

**STRING**

The JSON date-time element is processed as text.

**DEFAULT-ARRAY-MAXITEMS = value**

Specifies the maximum array boundary to apply where no maximum occurrence information (maxItems) is implied in the JSON schema. If this parameter is not set, no maximum limit is applied. The value of this parameter can be a positive integer in the range 1 - 2147483647. This parameter can be combined with use of the **INLINE-MAXOCCURS-LIMIT** parameter to influence how JSON arrays are mapped into the language structures.

**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 a JSON 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.

**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 | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

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 | 4.1 | 4.2 | 4.3 }**

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 region at CICS TS 5.3 or later when you want to use UTF-16.

## 4.1

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

## 4.2

For additional properties, use this mapping level with a region at CICS TS 5.4 or later.

## 4.3

For multidimensional array support, use this mapping level with a region at CICS TS 5.4 or later.

For more information about mapping levels, see [Mapping levels for the CICS assistants](#).

**MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL | [HYPHENS-AS-UNDERSCORES] | INTEGER-AS-PIC9 | LESS-DUP-NAMES | [UNDERSCORES-AS-HYPHENS] | [NO-ARRAY-NAME-INDEXING] }**

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

### **SAME-AS-MAPPING-LEVEL**

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

### **HYPHENS-AS-UNDERSCORES**

For PL/I only. This parameter converts any hyphens in the JSON schema to underscores rather than the character X, to improve the readability of the generated PL/I language structures. For more information, see [JSON schema to PL/I mapping](#). This option is enabled automatically at mapping level 4.2.

### **INTEGER-AS-PIC9**

For COBOL and DFHJS2LS only. This parameter generates language structures which contain integer values from the JSON schema as numerals rather than alphanumeric characters. This option is enabled automatically at mapping level 4.0

### **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 PL/I 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),
```

This option is enabled automatically at mapping level 4.2.

### **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](#).

This option is enabled automatically at mapping level 4.0.

### **NO-ARRAY-NAME-INDEXING**

For COBOL and 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 | 4.1 | 4.2 | 4.3 | 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 region.

#### **4.0**

The generated web service binding file deploys successfully into a region at CICS TS 5.3 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.

#### **4.1**

The generated web service binding file deploys successfully into a region at CICS TS 5.3 or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

#### **4.2**

The generated web service binding file deploys successfully into a region at CICS TS 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

### 4.3

The generated web service binding file deploys successfully into a region at CICS TS 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

#### **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 contain 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 a number to the prefix, with a maximum member name length of 8 characters.

#### **RESPMEM = value**

Specifies a one to six character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that contain 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 a number to the prefix, with a maximum member name length of 8 characters.

**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 | FULL | YES }**

Controls the maximum size of the packed decimal variable length in the generated COBOL or PL/I language structure.

**FULL**

For COBOL and PL/I. DFHJS2LS generates a packed decimal field that is large enough to hold all valid values. The maximum size is 31 digits. This is the default.

**NO**

For COBOL only. 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**

For COBOL only. DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

**Note:** The NO and YES options generate fields that are unable to represent all valid values; the FULL option avoids this problem. However, the FULL option does allow some invalid values to be represented in the packed decimal field. For example, if a schema indicates that there are a maximum of five digits and a maximum of two fractional digits, the FULL option will generate a packed decimal field that allows for seven digits, and this allows space for valid values such as 25000 and 999.99, but also provides space for some invalid values such as 9999.99. When you use the FULL option, take care not to generate invalid values in application data.

**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's 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. This topic lists the job control statements, symbolic parameters, input parameters and their descriptions for DFHJS2LS.

The DFHJS2LS JCL procedure is installed in the data set `HLQ .XDFHINST` , where `HLQ` is the high-level qualifier where CICS is installed.

**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.

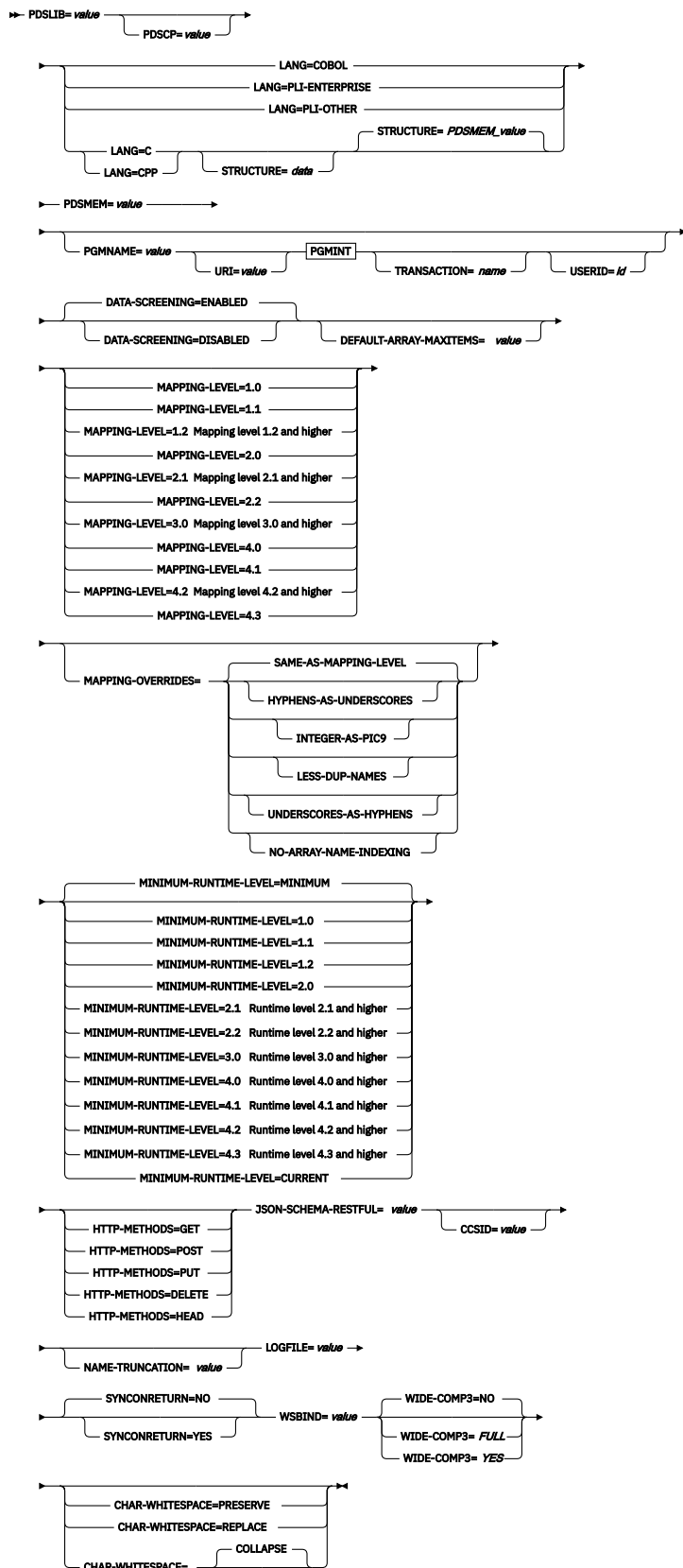
The default names for the files, when **TMPDIR** and **TMPFILE** are not specified, are as follows:

```
/tmp/JS2LS.in
/tmp/JS2LS.out
/tmp/JS2LS.err
```

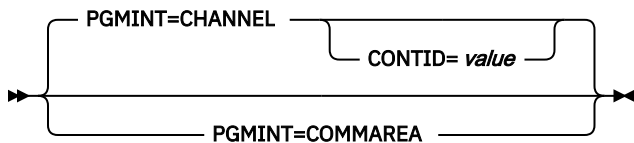
**Important:** DFHJS2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHJS2LS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading 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.

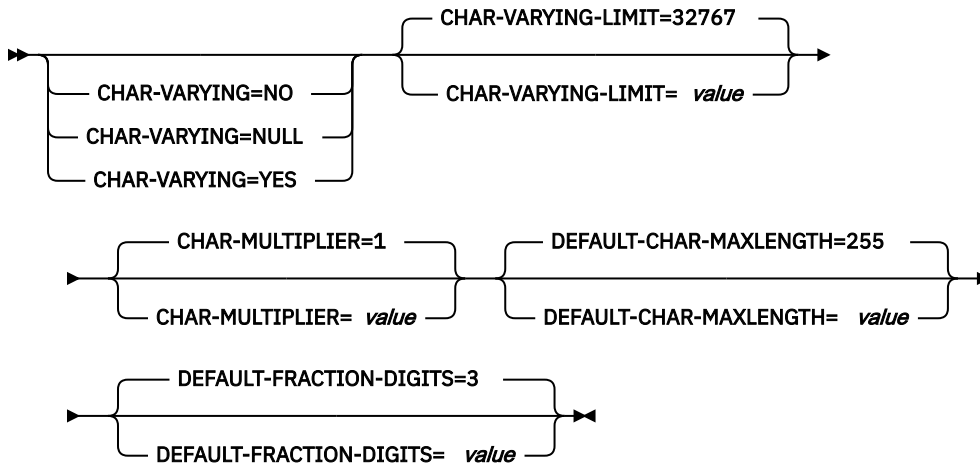
# Input parameters for DFHJS2LS



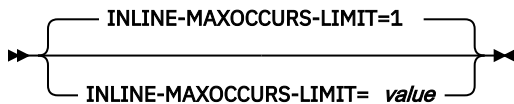
## PGMINT



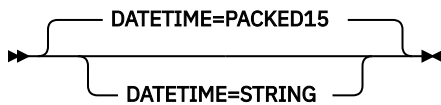
### Mapping level 1.2 and higher



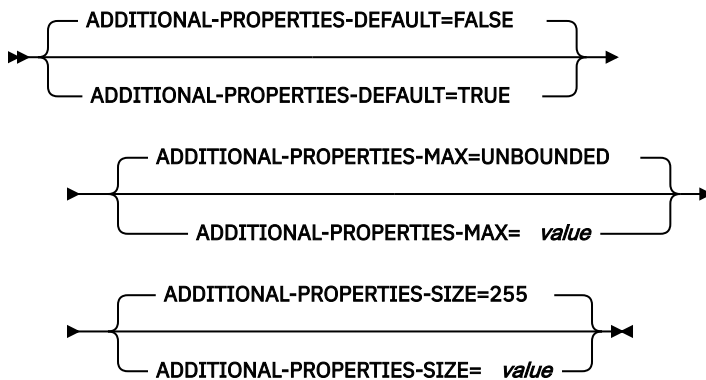
### Mapping level 2.1 and higher



### Mapping level 3.0 and higher



### Mapping level 4.2 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

### **ADDITIONAL-PROPERTIES-DEFAULT = { true | false }**

Indicates whether JSON schema objects that do not explicitly declare support for Additional Properties are interpreted as supporting them or not. Additional JSON properties are any properties within a JSON object that are not pre-defined in the JSON Schema. These properties are typically rejected by the data transformation mechanism as unexpected extra data. If **ADDITIONAL-PROPERTIES-DEFAULT** is set to TRUE, or if the JSON schema explicitly sets `additionalProperties:true` for an object, then space is allocated in the generated copybooks to hold such values. Applications can interact with those values using the associated fields in the copybooks.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-MAX = { 0-20 | UNBOUNDED }**

Indicates how many Additional Properties are supported for a JSON object that supports them. See **ADDITIONAL-PROPERTIES-DEFAULT**. The generated copybooks will contain structures suitable for addressing any additional properties. By default, there is no maximum constraint placed on the number of properties that are supported. The copybooks are generated in a similar fashion to arrays with no constraints and use containers. This parameter can be used to apply a maximum constraint that can be used in combination with the **INLINE-MAXOCCURS-LIMIT** parameter to cause a fixed length array to be allocated for the maximum number of properties, thereby avoiding the need for containers.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-SIZE = { 16-32767 | 255 }**

Indicates the maximum size for each of the JSON additional properties. If a JSON object supports additional properties, as defined by **ADDITIONAL-PROPERTIES-DEFAULT**, then the generated copybooks will have bindings to support properties up to the number specified by **ADDITIONAL-PROPERTIES-MAX**. By default, the maximum value supported for each additional property is 255 characters. A field of that size will be generated into the copybooks that are produced. This size can be customized by setting the **ADDITIONAL-PROPERTIES-SIZE** parameter. For example, a JSON object is processed that is found to contain the following property:

```
"example": { "notes": "this extra property was not defined in the JSON Schema" }
```

If the copybooks have been generated to support additional properties, then that entire value will be passed to the application for processing. The value begins with the leading quotation mark before the property's key, and ends with the trailing right brace in the property's value. It's approximately 100 characters in this example. The value used for **ADDITIONAL-PROPERTIES-SIZE** must be large enough to hold the largest such value that might occur. If the allocated buffer is too small for the value that is processed, an error response is generated.

You can use this parameter at a mapping level of 4.2 or higher.

### **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.

**CHAR-WHITESPACE = COLLAPSE | REPLACE | PRESERVE**

Specifies how white space in values of type string will be handled by CICS.

**COLLAPSE**

Leading, trailing, and embedded white space is removed and all tabs, new lines, and consecutive spaces are replaced with single space characters.

**REPLACE**

Any tabs or new lines are replaced with the appropriate number of spaces.

**PRESERVE**

Retains any white space in the data value.

If the **CHAR-WHITESPACE** parameter is not set, white space will be collapsed.

**Note:** This parameter does not apply to fields with a format of date-time , uri , base64Binary or hexBinary , where white space is always collapsed.

**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-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

**ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

**DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

**DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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-ARRAY-MAXITEMS = value**

Specifies the maximum array boundary to apply where no maximum occurrence information (maxItems) is implied in the JSON schema. If this parameter is not set, no maximum limit is applied. The value of this parameter can be a positive integer in the range 1 - 2147483647. This parameter can be combined with use of the **INLINE-MAXOCCURS-LIMIT** parameter to influence how JSON arrays are mapped into the language structures.

**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 a JSON 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.

**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 | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

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 | 4.1 | 4.2 | 4.3 }**

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 region at CICS TS 5.3 or later when you want to use UTF-16.

## 4.1

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

## 4.2

For additional properties, use this mapping level with a region at CICS TS 5.4 or later.

## 4.3

For multidimensional array support, use this mapping level with a region at CICS TS 5.4 or later.

For more information about mapping levels, see [Mapping levels for the CICS JSON assistants](#).

**MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL | [HYPHENS-AS-UNDERSCORES] | INTEGER-AS-PIC9 | LESS-DUP-NAMES | [UNDERSCORES-AS-HYPHENS] | [NO-ARRAY-NAME-INDEXING] }**  
Specifies whether the default behavior is overridden for the specified mapping level when generating language structures.

### **SAME-AS-MAPPING-LEVEL**

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

### **HYPHENS-AS-UNDERSCORES**

For PL/I only. This parameter converts any hyphens in the JSON schema to underscores rather than the character X, to improve the readability of the generated PL/I language structures. For more information, see [JSON schema to PL/I mapping](#). This option is enabled automatically at mapping level 4.2.

### **INTEGER-AS-PIC9**

For COBOL and DFHJS2LS only. This parameter generates language structures which contain integer values from the JSON schema as numerals rather than alphanumeric characters. This option is enabled automatically at mapping level 4.0

### **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 PL/I 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),
```

This option is enabled automatically at mapping level 4.2.

### **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](#).

This option is enabled automatically at mapping level 4.0.

### **NO-ARRAY-NAME-INDEXING**

For COBOL and 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 | 4.1 | 4.2 | 4.3 | 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 region.

#### **4.0**

The generated web service binding file deploys successfully into a region at CICS TS 5.3 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.

#### **4.1**

The generated web service binding file deploys successfully into a region at CICS TS 5.3 or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

## 4.2

The generated web service binding file deploys successfully into a region at CICS TS 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

## 4.3

The generated web service binding file deploys successfully into a region at CICS TS 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

## 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 names of the partitioned data set members that will contain the high-level language structures.

The member name is generated by appending a number to the prefix, with a maximum member name length of 8 characters.

## **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 *name 01* .

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 | FULL | YES }**

Controls the maximum size of the packed decimal variable length in the generated COBOL or PL/I language structure.

**FULL**

For COBOL and PL/I. DFHJS2LS generates a packed decimal field that is large enough to hold all valid values. The maximum size is 31 digits. This is the default.

**NO**

For COBOL only. 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**

For COBOL only. DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

**Note:** The NO and YES options generate fields that are unable to represent all valid values; the FULL option avoids this problem. However, the FULL option does allow some invalid values to be represented in the packed decimal field. For example, if a schema indicates that there are a maximum of five digits and a maximum of two fractional digits, the FULL option will generate a packed decimal field that allows for seven digits, and this allows space for valid values such as 25000 and 999.99, but also provides space for some invalid values such as 9999.99. When you use the FULL option, take care not to generate invalid values in application data.

**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's 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
/*
```

**DFHLS2JS: High-level language to JSON schema conversion for linkable interface**

The DFHLS2JS cataloged procedure generates a JSON schema and a JSON binding file from a high-level language structure. Use DFHLS2JS when you want to create a CICS program that can parse or create JSON.

The DFHLS2JS JCL procedure is installed in the data set `HLQ.XDFHINST`, where `HLQ` is the high-level qualifier where CICS is installed.

**Note:** **EXEC CICS TRANSFORM JSONTODATA** and **TRANSFORM DATATOJSON** commands supersede the linkable interface, DFHJSON, making it possible to convert JSON data to a language structure, or vice versa, in your application.

**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 specified in the input stream. However, you can also define them 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/ giving 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 that is 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 **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 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 DFHLS2JS uses to construct the names of the temporary workspace files.

The default value is LS2JS.

**PATHMAIN = path**

Specifies the name of the CICS TS directory in the UNIX system services file system. The value of this parameter is appended to the value specified by the **USSDIR** parameter.

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.

**USSDIR = path**

Specifies the name of the CICS TS directory in the z/OS UNIX file system. The value of this parameter is appended to the value specified by the **PATHMAIN** parameter. 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
```

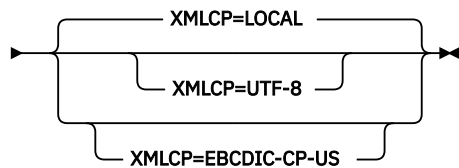
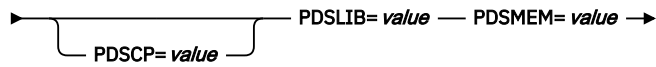
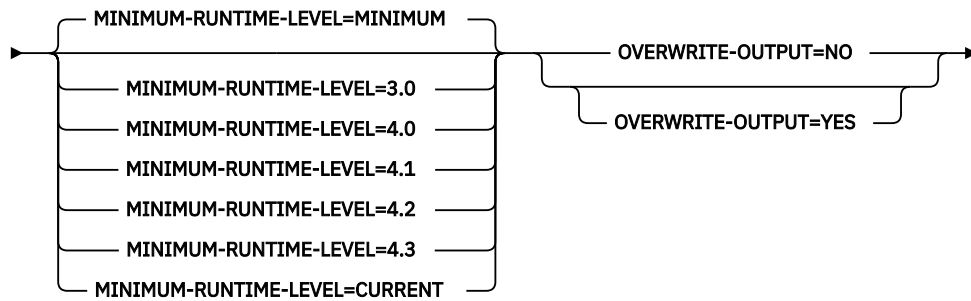
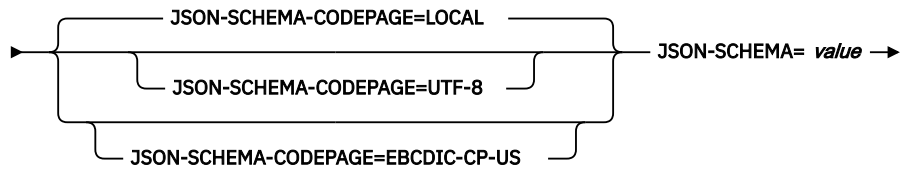
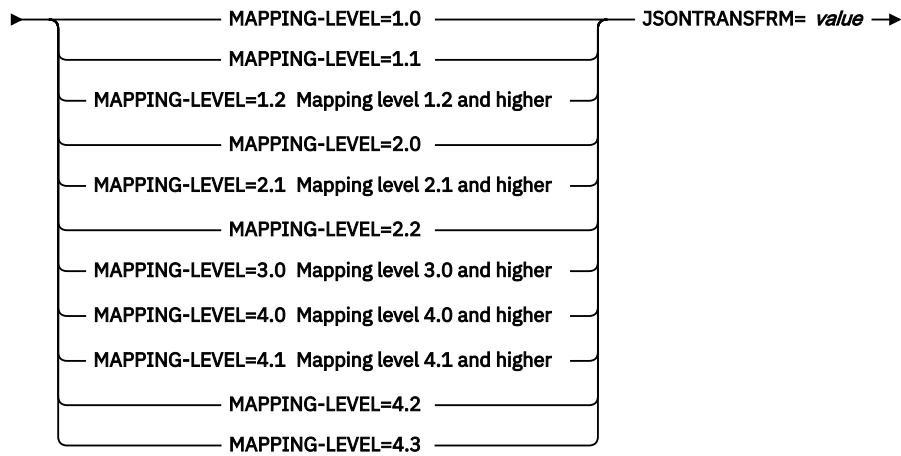
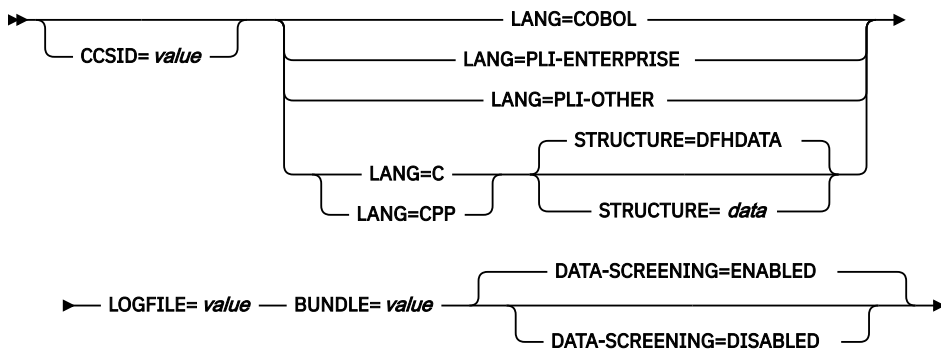
**Important:** DFHLS2JS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHLS2JS run concurrently and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading to unpredictable failures.

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.

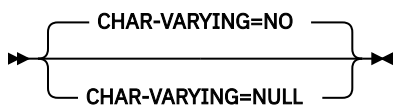
These temporary files are deleted before the end of the job.

### **Input parameters for DFHLS2JS**

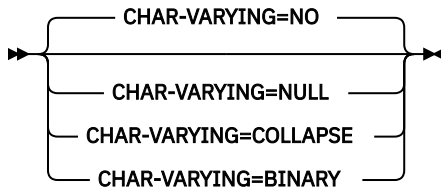
The following syntax diagram shows the available input parameters:



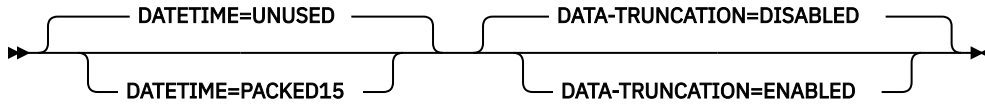
**Mapping level 1.2 and higher**



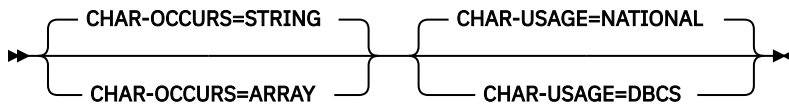
### Mapping level 2.1 and higher



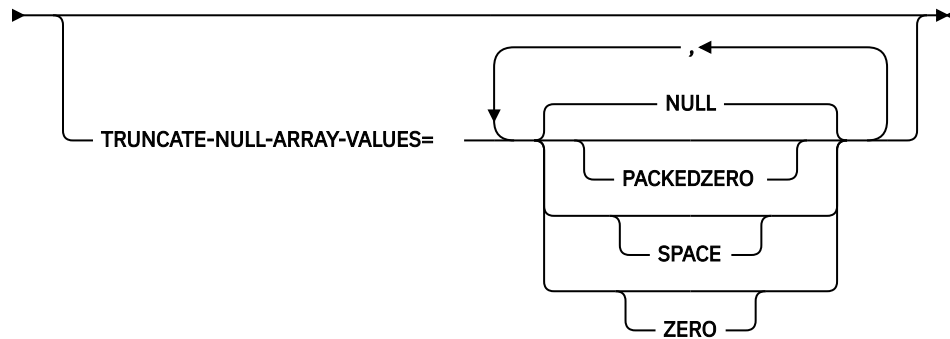
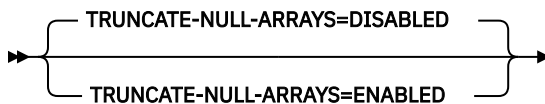
### Mapping level 3.0 and higher



### Mapping level 4.0 and higher



### Mapping level 4.1 and higher



## Parameter use

Parameters must conform to the following rules:

- 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.
- A number sign (#) 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

### **BUNDLE = value**

Specifies the path and name of the bundle directory on z/OS UNIX . If you specify this value, the JSON assistant generates a bundle that contains the JSON binding. The path information for this parameter overrides any path information for the **JSONTRANSFRM** parameter.

You can optionally specify an archive file rather than a directory name. The JSON assistant supports .zip and .jar archives. However, you must uncompress the archive before you try to install the BUNDLE resource.

### **CCSID = value**

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

You can use this parameter with any mapping level.

### **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. This parameter does not apply to Enterprise and Other PL/I language structures. 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.

#### **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 schema. 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++.

#### **COLLAPSE**

Character fields are mapped to a JSON string. Trailing and embedded white space in the field is not included in the JSON message; for example <space>AB<space><space><space>C<space> becomes AB<space>C. 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 greater.

### **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.

#### **DATA-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

##### **ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

##### **DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

#### **DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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 JSON date-time properties in the high-level language structure, including CICS ABSTIME values, are mapped as time stamps:

##### **PACKED15**

Any JSON date-time properties are mapped as time stamps.

##### **UNUSED**

Any JSON date-time properties are not mapped as time stamps. This mapping is the default.

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

#### **JSON-SCHEMA = *value***

The fully qualified z/OS UNIX name of the file into which the JSON schema is written.

#### **JSON-SCHEMA-CODEPAGE = { LOCAL | UTF-8 | EBCDIC-CP-US }**

Specifies the code page that is used to generate the JSON Schema documents.

##### **LOCAL**

Specifies that the JSON Schemas are generated using the default code page for the file system.

##### **UTF-8**

Specifies that the JSON Schemas are generated using the UTF-8 code page.

##### **EBCDIC-CP-US**

Specifies that the JSON Schemas are generated using the US EBCDIC code page.

#### **JSONTRANSFRM = *value***

This parameter is mandatory for LINKable mode, but invalid for request-response mode. It indicates the name that is used for the **JSONTRANSFRM** bundle resource in CICS.

#### **LANG = COBOL | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

C++

**LOGFILE = value**

The fully qualified z/OS UNIX name of the file into which DFHLS2JS writes its activity log and trace information. If it does not exist, DFHLS2JS creates the file, but not the directory structure.

You might be requested to use this parameter 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 | 4.1 | 4.2 | 4.3 }**

Specifies the level of mapping for the assistant to use when generating the JSON binding and language structures. You must use mapping level 3.0 or higher.

**MINIMUM-RUNTIME-LEVEL = { MINIMUM | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | CURRENT }**

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

**MINIMUM**

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

**3.0**

Specify runtime level 3.0 or above if you want to use the CICS JSON assistant and take advantage of advanced data mappings.

**4.0**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, 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.

**4.1**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

**4.2**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

**4.3**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

**CURRENT**

Use this runtime level to deploy the generated binding file into a CICS region that has the same runtime environment as the region used to generate the binding file.

**OVERWRITE-OUTPUT = { NO | YES }**

Controls whether existing CICS BUNDLES on the file system can be overwritten.

**NO**

Any existing BUNDLE is not replaced. If an existing BUNDLE is found DFHLS2JS issues error message DFHPI9689E and terminates.

**YES**

Any existing BUNDLE is replaced. If an existing BUNDLE is found then message DFHPI9683W is issued to inform you that the file has been replaced.

**PDSCP = value**

Specifies the code page that is used in the partitioned data set members, where *value* is a CCSID number or a Java code page number. If you do not specify this parameter, 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 high-level language data structures to be processed.

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

**PDSMEM = value**

Specifies the name of the partitioned data set member that contains the high-level language structures that you want to process.

**STRUCTURE = { DFHDATA | data }**

The name of the top-level data structure in C and C++. The default is DFHDATA.

**TRUNCATE-NULL-ARRAYS = { DISABLED | ENABLED }**

Specifies how structured arrays are processed at mapping level 4.1 or higher. If enabled, CICS will attempt to recognize empty records within an array (see TRUNCATE-NULL-ARRAY-VALUES for more information about identifying empty records). If five consecutive empty array records are detected, the array is truncated at the first such record when generating XML/JSON. This truncation capability is only enabled for arrays with structured content, arrays of simple primitive fields are not subject to truncation. Truncation of arrays can result in a more concise representation of the data in JSON/XML, but is not without risk. If five consecutive data records are misidentified as uninitialized storage (perhaps because they legitimately contain low values), data loss can be experienced. If TRUNCATE-NULL-ARRAYS is enabled and TRUNCATE-NULL-ARRAY-VALUES is not set, then the default value for TRUNCATE-NULL-ARRAY-VALUES is used.

**TRUNCATE-NULL-ARRAY-VALUES = { NULL | PACKEDZERO | SPACE | ZERO }**

Specifies which values are treated as empty for TRUNCATE-NULL-ARRAYS processing at mapping level 4.1 or higher. By default, the null value (0x00, or low-values) is treated as empty. If all of the bytes of storage within a record of a structured array contain nulls, then the entire record is considered to be empty. One or more of the NULL, PACKEDZERO, SPACE and ZERO values can be specified in a comma separated list.

**NULL**

Implies a null character (0x00).

**PACKEDZERO**

Implies a positive signed packed decimal zero (0x0C), a negative signed packed decimal zero (0x0D), or an unsigned packed decimal zero (0x0F).

**SPACE**

Implies an SBCS EBCDIC space (0x40).

**ZERO**

Implies an unsigned zoned decimal zero (0xF0).

Any matching combination of the selected bytes within a structured array record will cause the entire record to be identified as empty.

If TRUNCATE-NULL-ARRAY-VALUES has a value defined, TRUNCATE-NULL-ARRAYS must be enabled.

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
// SET QT=' '
//JAVAPROG EXEC DFHLS2JS,
```

```
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/example.log
MAPPING-LEVEL=4.0
JSONTRANSFRM=EXAMPLE
BUNDLE=/u/exampleapp/bundles/exampleBundle
/*
```

## DFHJS2LS: JSON schema to high-level language conversion for linkable interface

The DFHJS2LS cataloged procedure generates a high-level language data structure and a JSON binding from a JSON schema. Use DFHJS2LS when you want to create a CICS program that can parse or create JSON. This topic lists the job control statements, symbolic parameters, input parameters and their descriptions for DFHJS2LS.

The DFHJS2LS JCL procedure is installed in the data set *HLQ.XDFHINST*, where *HLQ* is the high-level qualifier where CICS is installed.

**Note:** **EXEC CICS TRANSFORM JSONTODATA** and **TRANSFORM DATATOJSON** commands supersede the linkable interface, DFHJSON, making it possible to convert JSON data to a language structure, or vice versa, in your application.

### 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 specified in the input stream. You can also define them 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/* giving 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 that is 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 **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.

**PATHMAIN =*path***

Specifies the main part of the name of the CICS TS directory in the UNIX system services file system.

The default value is /usr/lpp/cicsts.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHMAIN** parameter.

**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 the value specified by the **PATHMAIN** parameter. 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

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:

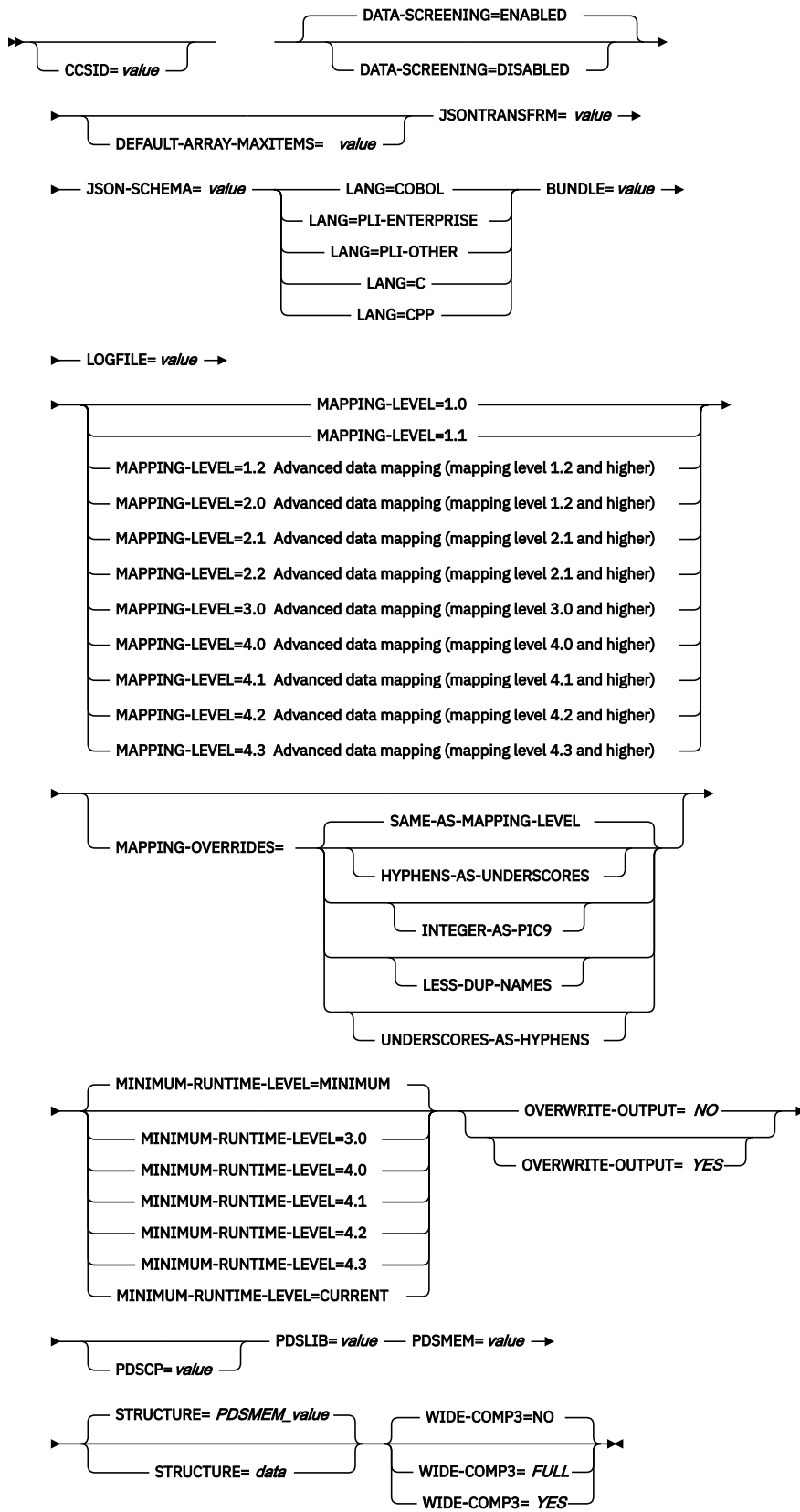
```
/tmp/JS2LS.in  
/tmp/JS2LS.out  
/tmp/JS2LS.err
```

**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.

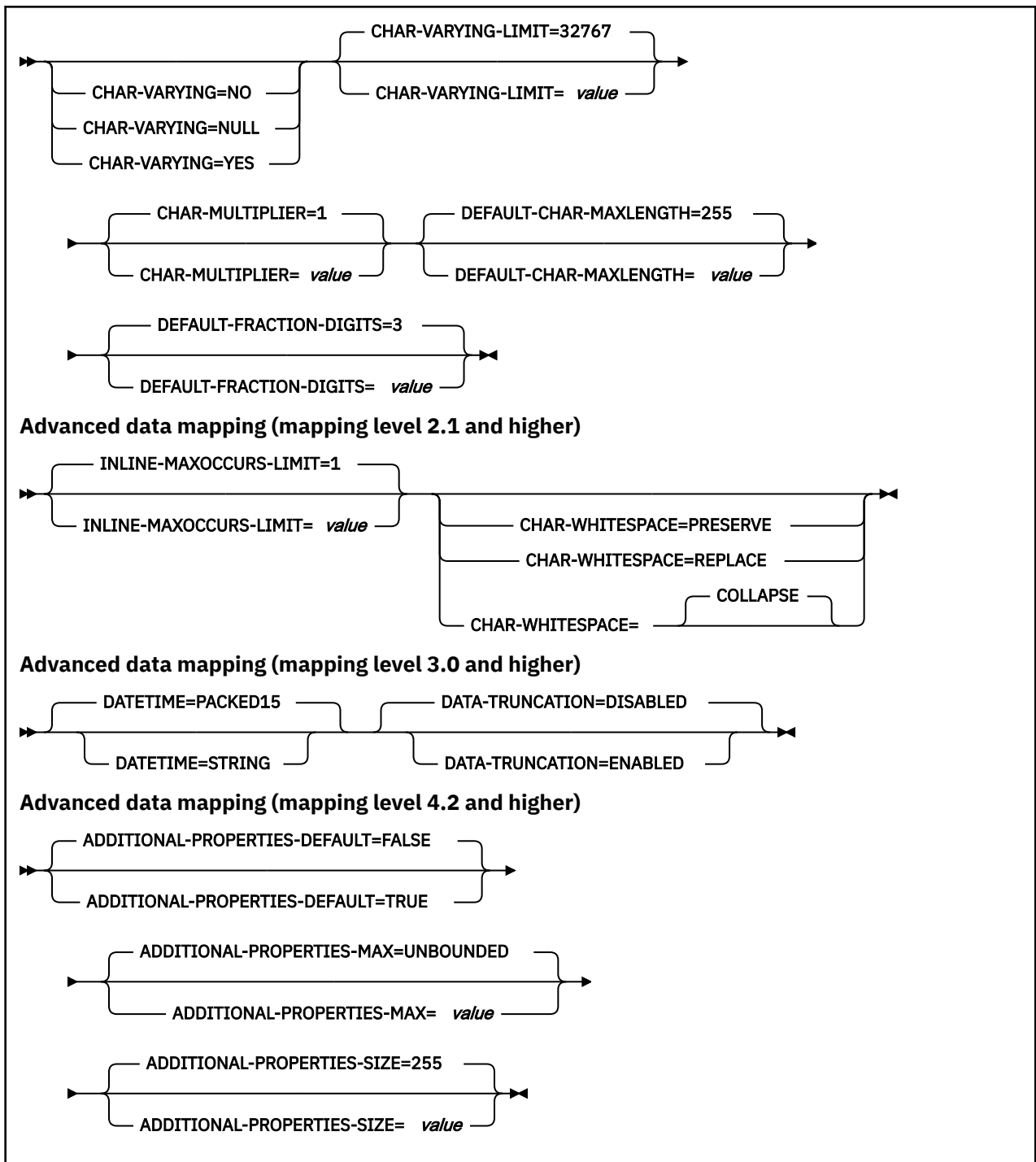
Therefore, you are advised to devise a naming convention and operating procedures that avoid this situation; for example, you can use the system symbolic parameter **SYSUID** to generate workspace file names that are unique to an individual user.

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

## Input parameters for DFHJS2LS



Advanced data mapping (mapping level 1.2 and higher)



## 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.
- 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

### **ADDITIONAL-PROPERTIES-DEFAULT = { true | false }**

Indicates whether JSON schema objects that do not explicitly declare support for Additional Properties are interpreted as supporting them or not. Additional JSON properties are any properties within a JSON object that are not pre-defined in the JSON Schema. These properties are typically rejected by the data transformation mechanism as unexpected extra data. If **ADDITIONAL-PROPERTIES-DEFAULT** is set to TRUE, or if the JSON schema explicitly sets `additionalProperties:true` for an object, then space is allocated in the generated copybooks to hold such values. Applications can interact with those values using the associated fields in the copybooks.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-MAX = { 0-20 | UNBOUNDED }**

Indicates how many Additional Properties are supported for a JSON object that supports them. See **ADDITIONAL-PROPERTIES-DEFAULT**. The generated copybooks will contain structures suitable for addressing any additional properties. By default, there is no maximum constraint placed on the number of properties that are supported. The copybooks are generated in a similar fashion to arrays with no constraints and use containers. This parameter can be used to apply a maximum constraint that can be used in combination with the **INLINE-MAXOCCURS-LIMIT** parameter to cause a fixed length array to be allocated for the maximum number of properties, thereby avoiding the need for containers.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-SIZE = { 16-32767 | 255 }**

Indicates the maximum size for each of the JSON additional properties. If a JSON object supports additional properties, as defined by **ADDITIONAL-PROPERTIES-DEFAULT**, then the generated copybooks will have bindings to support properties up to the number specified by **ADDITIONAL-PROPERTIES-MAX**. By default, the maximum value supported for each additional property is 255 characters. A field of that size will be generated into the copybooks that are produced. This size can be customized by setting the **ADDITIONAL-PROPERTIES-SIZE** parameter. For example, a JSON object is processed that is found to contain the following property:

```
"example": { "notes": "this extra property was not defined in the JSON Schema" }
```

If the copybooks have been generated to support additional properties, then that entire value will be passed to the application for processing. The value begins with the leading quotation mark before the property's key, and ends with the trailing right brace in the property's value. It's approximately 100 characters in this example. The value used for **ADDITIONAL-PROPERTIES-SIZE** must be large enough to hold the largest such value that might occur. If the allocated buffer is too small for the value that is processed, an error response is generated.

You can use this parameter at a mapping level of 4.2 or higher.

### **BUNDLE = value**

Specifies the path and name of the bundle directory on z/OS UNIX. If you specify this value, the JSON assistant generates the JSON binding in the bundle directory and creates a bundle manifest for you. The path information for this parameter overrides any path information for the **JSONTRANSFRM** parameter.

### **CCSID = value**

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

You can use this parameter with any mapping level.

**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 consists of two related elements: the 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.

**CHAR-WHITESPACE = COLLAPSE | REPLACE | PRESERVE**

Specifies how white space in values of type string will be handled by CICS.

**COLLAPSE**

Leading, trailing, and embedded white space is removed and all tabs, new lines, and consecutive spaces are replaced with single space characters.

**REPLACE**

Any tabs or new lines are replaced with the appropriate number of spaces.

**PRESERVE**

Retains any white space in the data value.

If the **CHAR-WHITESPACE** parameter is not set, white space will be collapsed.

**Note:** This parameter does not apply to fields with a format of date-time , uri , base64Binary or hexBinary , where white space is always collapsed.

**DATA-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

**ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

**DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

**DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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 that JSON date-time properties are mapped to CICS ABSTIME data format or to text:

**PACKED15**

JSON date-time properties fields are mapped to CICS ABSTIME format.

**STRING**

JSON date-time properties are mapped to text. This mapping is the same as all previous mappings levels.

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

**DEFAULT-ARRAY-MAXITEMS = *value***

Specifies the maximum array boundary to apply where no maximum occurrence information (maxItems) is implied in the JSON schema. If this parameter is not set, no maximum limit is applied. The value of this parameter can be a positive integer in the range 1 - 2147483647. This parameter can be combined with use of the **INLINE-MAXOCCURS-LIMIT** parameter to influence how JSON arrays are mapped into the language structures.

**DEFAULT-CHAR-MAXLENGTH = { 255 | *value* }**

Specifies the default array length of character data in characters for mappings where no length is implied in the JSON schema document, when the mapping level is 1.2 or later. The *value* of this parameter can be a positive integer in the range of 1 to 2 147 483 647.

**DEFAULT-FRACTION-DIGITS = { 3 | *value* }**

Specifies the default number of fraction digits to use on a JSON 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.

**INLINE-MAXOCCURS-LIMIT = { 1 | *value* }**

Specifies whether inline variable repeating content is used based on the maxItems attribute of the JSON schema keywords.

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 to 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 maxItems attribute is greater than the *value* of **INLINE-MAXOCCURS-LIMIT**, container-based mapping is used; otherwise, inline mapping is used.

If you decide that you want to variably repeat lists to be mapped inline, consider the length of a single item of recurring data. If you have few instances of long length, container-based mapping is preferable; if you have many instances of short length, inline mapping is likely to be preferable.

**JSONTRANSFRM = value**

This parameter is mandatory for LINKable mode, but invalid for request-response and RESTful modes. It indicates the name (1 - 16 characters) that is used for the **JSONTRANSFRM** bundle-resource in CICS. You can use the characters A - Z a - z 0 - 9 and \_ in the name.

**JSON-SCHEMA = value**

The fully qualified z/OS UNIX name of the file from which the JSON schema is read. If it does not already exist, DFHJS2LS creates the file, but not the directory structure.

**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. If it does not already exist, DFHJS2LS creates the file, but not the directory structure.

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 | 4.1 | 4.2 | 4.3 }**

Specifies the level of mapping for the assistant to use when generating the JSON binding and language structures. You must use the most recent mapping level that is available; for DFHJS2LS, you must use a mapping level of 3.0 or higher.

**3.0**

This is the minimum mapping level you can use with DFHJS2LS.

**4.0**

Use this mapping level with a region at CICS TS 5.3 or later when you want to use UTF-16.

**4.1**

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

**4.2**

For additional properties, use this mapping level with a region at CICS TS 5.4 or later.

**4.3**

For multidimensional array support, use this mapping level with a region at CICS TS 5.4 or later.

**MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL | [HYPHENS-AS-UNDERSCORES] | INTEGER-AS-PIC9 | LESS-DUP-NAMES | [UNDERSCORES-AS-HYPHENS] | [NO-ARRAY-NAME-INDEXING] }**

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

**SAME-AS-MAPPING-LEVEL**

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

**HYPHENS-AS-UNDERSCORES**

For PL/I only. This parameter converts any hyphens in the JSON schema to underscores rather than the character X, to improve the readability of the generated PL/I language structures. For more information, see [JSON schema to PL/I mapping](#). This option is enabled automatically at mapping level 4.2.

## INTEGER-AS-PIC9

For COBOL and DFHJS2LS only. This parameter generates language structures which contain integer values from the JSON schema as numerals rather than alphanumeric characters. This option is enabled automatically at mapping level 4.0

## 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 PL/I 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),
```

This option is enabled automatically at mapping level 4.2.

## 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](#).

This option is enabled automatically at mapping level 4.0.

## NO-ARRAY-NAME-INDEXING

For COBOL and 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** | **3.0** | **4.0** | **4.1** | **4.2** | **4.3** | **CURRENT** }

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

### MINIMUM

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

### 3.0

Specify runtime level 3.0 or above if you want to use the CICS JSON assistant and take advantage of advanced data mappings.

### 4.0

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, 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.

### 4.1

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

### 4.2

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

### 4.3

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

#### **CURRENT**

Use this runtime level to deploy the generated binding file into a CICS region that has the same runtime environment as the region used to generate the binding file.

#### **OVERWRITE-OUTPUT = { NO | YES }**

Controls whether existing CICS BUNDLES on the file system can be overwritten.

#### **NO**

Any existing BUNDLE is not replaced. If an existing BUNDLE is found DFHJS2LS issues error message DFHPI9689E and terminates.

#### **YES**

Any existing BUNDLE is replaced. If an existing BUNDLE is found then message DFHPI9683W is issued to inform you that the file has been replaced.

#### **PDSCP = value**

Specifies the code page that is used in the partitioned data set members, where *value* is a CCSID number or a Java code page number. If you do not specify this parameter, 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.

#### **PDSMEM = value**

Specifies the 1-6 character prefix that DFHJS2LS uses to generate the names of the partitioned data set members that will contain the high-level language structures.

DFHJS2LS generates a partitioned data set member for each operation. It generates the member name by appending a number to the prefix, with a maximum member name length of 8 characters.

#### **STRUCTURE = { PDSMEM\_value | data }**

The name of the top-level data structure in C and C++. The default value is the value of the **PDSMEM** parameter.

#### **WIDE-COMP3 = { NO | FULL | YES }**

Controls the maximum size of the packed decimal variable length in the generated COBOL or PL/I language structure.

#### **FULL**

For COBOL and PL/I. DFHJS2LS generates a packed decimal field that is large enough to hold all valid values. The maximum size is 31 digits. This is the default.

#### **NO**

For COBOL only. 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**

For COBOL only. DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

**Note:** The NO and YES options generate fields that are unable to represent all valid values; the FULL option avoids this problem. However, the FULL option does allow some invalid values to be represented in the packed decimal field. For example, if a schema indicates that there are a maximum of five digits and a maximum of two fractional digits, the FULL option will generate a packed decimal field that allows for seven digits, and this allows space for valid values such as 25000 and 999.99, but also provides space for some invalid values such as 9999.99. When you use the FULL option, take care not to generate invalid values in application data.

## 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=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
JSONTRANSFRM=EXAMPLE
BUNDLE=/u/exampleapp/bundles/exampleBundle
/*
```

## How the CICS JSON assistant maps between high-level languages and JSON schemas

Utility programs DFHJS2LS and DFHLS2JS are collectively known as the CICS JSON assistant. The CICS JSON assistant generates mappings between high-level language structures and JSON schemas. The assistant also generates JSON schemas from high-level language data structures, or vice versa.

- DFHLS2JS maps high-level language structures to JSON schemas.
- DFHJS2LS maps JSON schemas to high-level language structures.

The two mappings are not symmetrical:

- If you process a language data structure with DFHLS2JS, and then process the resulting JSON schema with the DFHJS2LS, do not expect the final data structure to be the same as the original.
- If you process a JSON schema with DFHJS2LS, and then process the resulting language structure with the DFHLS2JS, do not expect the final JSON schema to be the same as the original.
- In some cases, DFHJS2LS generates language structures that are not supported by DFHLS2JS.

You must code high-level language structures that are processed by DFHLS2JS according to the rules of the language, as implemented in the language compilers that CICS supports.

### Mapping levels for the CICS JSON assistant

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

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

### Limitations at all mapping levels

- Data types used in the JSON schema must be explicitly declared.
- JSON object references to external documents using `$ref` are not supported within the JSON schema.

### Mapping level 4.3

Mapping level 4.3 is compatible with CICS TS V5.4 and higher.

Mapping level 4.3 is primarily for use with DFHJS2LS, but is also included in the CICS web services assistants, XML assistants, and JSON assistants. This mapping level implements support for multidimensional arrays in JSON.

## Mapping level 4.2

Mapping level 4.2 is compatible with CICS TS V5.4 and higher.

Mapping level 4.2 is primarily for use with DFHJS2LS, but is also included in the CICS web services assistants, XML assistants, and JSON assistants. This mapping level implements support for Additional Properties in JSON, and introduces the following three parameters to DFHJS2LS: **ADDITIONAL-PROPERTIES-DEFAULT**, **ADDITIONAL-PROPERTIES-MAX**, and **ADDITIONAL-PROPERTIES-SIZE**.

## Mapping level 4.1

Mapping level 4.1 is compatible with a CICS TS 5.3 region, and higher.

Mapping level 4.1 is added to the CICS web services assistants, XML assistants, and JSON assistants. This mapping level implements improved mappings for simple arrays generated bottom-up from existing copybooks; it also adds the ability for CICS to auto-detect uninitialized trailing storage in arrays, and to omit those records from the generated XML/JSON form.

DFHLS2WS, DFHWS2LS, DFHLS2SC, DFHSC2LS, DFHJS2LS, and DFHLS2JS support the **TRUNCATE-NULL-ARRAYS** and **TRUNCATE-NULL-ARRAY-VALUES** parameters.

If you specify any value for **TRUNCATE-NULL-ARRAY-VALUES**, you must also specify **TRUNCATE-NULL-ARRAYS=ENABLED**.

## Mapping level 4.0

This mapping level provides the following support:

At mapping level 4.0 and higher, DFHLS2JS supports the COBOL OCCURS DEPENDING ON clause and supports mapping of COBOL character arrays into JSON schemas. You can set this behavior by using the **CHAR-OCCURS** parameter on the CICS JSON assistant.

- You must specify the parameter **DATA-TRUNCATION=ENABLED**.
- Complex OCCURS DEPENDING ON is not supported. This limitation means that OCCURS DEPENDING ON is only supported for the last field of a structure.
- CICS does not support qualified names (using the 'OF' keyword) as the target of an OCCURS DEPENDING ON clause, for example FIELD1 OF STRUCTURE1.
- CICS does not support the UNBOUNDED keyword. You must specify an integer bound for the maximum size of the table expected by the application.

At mapping level 4.0 and higher, JSON web services support the conversion of application data that is encoded using UTF-16 Unicode.

- When you use LS2JS, you can enable this behavior by using language-specific data types for UTF-16.
- When you use JS2LS, you can enable this behavior by setting CCSID=1200.
- CICS supports only a single Unicode code page, " UTF-16BE with IBM Private Use Area " (CCSID 1200).
- Conversion of application data that is encoded using UTF-8 is not supported.

**Note:** DFHLS2JS does not support the COBOL GROUP USAGE NATIONAL clause.

## Mapping level 3.0 and higher

This mapping level provides the following support:

- DFHJS2LS maps date-time data types to CICS ASKTIME format.
- DFHLS2JS can generate a JSON schema 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 assistant.

## Mapping level 2.2 and higher

This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

## Mapping level 2.1 and higher

This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

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, DFHJS2LS offers the following new and improved support for arrays:

- The **INLINE-MAXOCCURS-LIMIT** parameter
- The `minItems` property

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 in DFHJS2LS”](#) on page 296.

At mapping level 2.1 and higher, DFHLS2JS support the following JSON 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 JSON 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 a JSON schema. This option is available only on SBCS character arrays and allows the array to be mapped to a fixed-length JSON string containing base64 encoded data rather than to a normal string.

## Mapping level 1.2 and higher

This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

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 DFHJS2LS, note that the following rules apply after the value of this parameter is used to calculate the amount of space required for character data.

- DFHJS2LS provides 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 JSON schema that do not have a length associated with them, you can assign a default length using the **DEFAULT-CHAR-MAXLENGTH** parameter in DFHJS2LS.

DFHLS2JS provides these mappings:

- Character fields map to a `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

This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

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

- 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 JSON schema. 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. Validation has a significant performance impact.

### Mapping level 1.1 only

This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

This mapping level provides improved mapping of JSON character and binary data types, in particular when mapping data of variable length that has `maxLength` and `minLength` properties defined with different values in the JSON 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 JSON 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 JSON schema.
  - $\text{ceil}(x)$  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

This option is retained for compatibility with SOAP web services. It is not recommended for use with JSON.

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

- DFHJS2LS map character and binary data types in the JSON schema to fixed-length fields in the language structure. Look at this partial JSON schema:

```
"example":{
  "type":"string",
  "maxLength":33000}
```

That partial JSON schema appears in a COBOL language structure as:

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

- CICS encodes and decodes data in the hexBinary format but not in base64Binary format. DFHJS2LS map Base64Binary data to a fixed-length character field, the contents of which must be encoded or decoded by the application program.
- DFHLS2JS interpret character and binary fields in the language structure as fixed-length fields and map those fields to JSON strings that have a maxLength property. At run time, the fields in the language structure are filled with spaces or nulls if insufficient data is available.

## COBOL to JSON schema mapping

The DFHLS2JS utility program supports mappings between COBOL data structures and JSON schema definitions.

### How COBOL names are converted to JSON

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

- Duplicate names are made unique by the addition of one or more numeric digits.
  - For example, two instances of year become year and year1.
- Hyphens are replaced by underscores. Strings of contiguous hyphens are replaced by contiguous underscores.
  - For example, current-user--id becomes current\_user\_\_id.
- 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.
- A leading underscore is added to names that start with a numeric character.
  - For example, 9A-REQUEST-ID becomes \_9a\_request\_id.

## How COBOL data description elements are mapped to JSON

CICS maps COBOL data description elements to schema elements according to [Table 8 on page 259](#).

### Restriction:

- COBOL data description elements that are not shown in [Table 8 on page 259](#) are not supported by DFHLS2JS.
- 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:
  - 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 DFHLS2JS 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 that are supported for DISPLAY and COMPUTATIONAL-5 data description items are 9, S, and Z.
- The PICTURE characters that are supported for PACKED-DECIMAL data description items are 9, S, V, and Z.
- The only PICTURE characters that are 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 a 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 a string 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.
- The OCCURS DEPENDING ON clause is supported at a mapping level of 4.0 or higher. Complex OCCURS DEPENDING ON is not supported. This means that OCCURS DEPENDING ON is only supported for the last field of a structure.
- The OCCURS INDEXED BY clause is supported at any mapping level.

Table 8. Mapping reference for COBOL data description elements

COBOL data description	JSON schema definition
PIC X( <i>n</i> ) PIC A( <i>n</i> ) PIC G( <i>n</i> ) DISPLAY-1 PIC N( <i>n</i> )	<pre>"type": "string", "maxLength": <i>n</i></pre>
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY PIC S9( <i>n</i> ) DISPLAY PIC S9( <i>n</i> ) COMP PIC S9( <i>n</i> ) COMP-4 PIC S9( <i>n</i> ) COMP-5 PIC S9( <i>n</i> ) BINARY	<pre>"type": "integer", "minimum": -(<i>n</i>+1), "maximum": <i>n</i></pre> <p>where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY PIC 9( <i>n</i> ) DISPLAY PIC 9( <i>n</i> ) COMP PIC 9( <i>n</i> ) COMP-4 PIC 9( <i>n</i> ) COMP-5 PIC 9( <i>n</i> ) BINARY	<pre>"type": "integer", "minimum": 0, "maximum": <i>n</i></pre> <p>where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC S9( <i>m</i> )V9( <i>n</i> ) COMP-3	<pre>"type": "number", "format": "decimal", "minimum": <i>x</i>, "maximum": <i>y</i>, "multipleOf": <i>z</i></pre> <p>where:</p> <p><i>x</i> is the minimum value that can be represented by the pattern of '9' characters.</p> <p><i>y</i> is the maximum value that can be represented by the pattern of '9' characters.</p> <p><i>z</i> is the smallest unit available = 1 / 10<sup><i>n</i></sup>.</p>
PIC 9( <i>m</i> )V9( <i>n</i> ) COMP-3	<pre>"type": "number", "format": "decimal", "minimum": 0, "maximum": <i>y</i>, "multipleOf": <i>z</i></pre> <p>where:</p> <p><i>y</i> is the maximum value that can be represented by the pattern of '9' characters.</p> <p><i>z</i> is the smallest unit available = 1 / 10<sup><i>n</i></sup>.</p>

Table 8. Mapping reference for COBOL data description elements (continued)

COBOL data description	JSON schema definition
<p>PIC S9(15) COMP-3</p> <p>Supported at mapping level 3.0 when DATETIME=PACKED15</p>	<pre data-bbox="836 254 1089 302">"type": "string", "format": "date-time"</pre> <p data-bbox="820 331 1450 363">The format of the time stamp is defined by <a href="#">RFC3339</a>.</p>
<p>PIC S9(m)V9(n) DISPLAY</p> <p>Supported at mapping level 1.2 and higher</p>	<pre data-bbox="836 512 1073 623">"type": "number", "format": "decimal", "minimum": x, "maximum": y, "multipleOf": z</pre> <p data-bbox="820 653 902 680">where:</p> <p data-bbox="820 699 1466 762">x is the minimum value that can be represented by the pattern of '9' characters.</p> <p data-bbox="820 781 1466 844">y is the maximum value that can be represented by the pattern of '9' characters.</p> <p data-bbox="820 863 1300 890">z is the smallest unit available = <math>1 / 10^n</math>.</p>
<p>COMP-1</p> <p>Supported at mapping level 1.2 and higher</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values might not convert exactly from one representation to the other.</p> <p>Some extremely large or small values might not be valid for float data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-1 data types with fixed precision alternatives.</p>	<pre data-bbox="836 942 1040 991">"type": "number", "format": "float"</pre>

Table 8. Mapping reference for COBOL data description elements (continued)

COBOL data description	JSON schema definition
<p>COMP-2</p> <p>Supported at mapping level 1.2 and higher</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values might not convert exactly from one representation to the other.</p> <p>Some extremely large or small values might not be valid for double data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-2 data types with fixed precision alternatives.</p>	<pre>"type": "number", "format": "double"</pre>

Table 8. Mapping reference for COBOL data description elements (continued)

COBOL data description	JSON schema definition
<p><i>data description</i> OCCURS <i>n</i> TIMES</p>	<p><b>At mapping level 4.0 and lower</b></p> <p>For primitives:</p> <pre data-bbox="865 321 1458 552"> " type": "array" " maxItems": n " minItems": n " items": {   " type": "object",   " properties": { name: { data description JSON }}   " required": [name] } </pre> <p>For structures:</p> <pre data-bbox="865 615 1458 720"> " type": "array" " maxItems": n " minItems": n " items": { data description JSON } </pre> <p>Where <i>data description JSON</i> is the JSON schema representation of the COBOL <i>data description</i> and <i>name</i> is the name of the COBOL <i>data description</i>.</p> <p><b>At mapping level 4.1 and higher</b></p> <p><b>TRUNCATE-NULL-ARRAYS = DISABLED</b></p> <p>Both structured and primitive arrays map as follows.</p> <pre data-bbox="865 1035 1458 1140"> " type": "array" " maxItems": n " minItems": n " items": { data description JSON } </pre> <p><b>TRUNCATE-NULL-ARRAYS = ENABLED</b></p> <p>Primitive arrays map as above, and structured arrays as follows.</p> <pre data-bbox="865 1287 1458 1392"> " type": "array" " maxItems": n " minItems": 0 " items": { data description JSON } </pre>
<p><i>data description</i> OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDING ON <i>t</i></p> <p>Supported at mapping level 4.0</p>	<pre data-bbox="833 1455 1458 1602"> " field-name": {   " type": "array",   " maxItems": m   " minItems": n   " items": { ... } } </pre> <p>The content of the array item depends on the datatype used.</p>

Table 8. Mapping reference for COBOL data description elements (continued)

COBOL data description	JSON schema definition
<p>PIC X OCCURS <i>n</i> TIMES                      PIC A OCCURS <i>n</i> TIMES                      PIC G DISPLAY-1 OCCURS <i>n</i> TIMES                      PIC N OCCURS <i>n</i> TIMES</p>	<p>When <b>CHAR-OCCURS</b> =STRING :</p> <pre data-bbox="831 296 1459 415">                     "field-name":{                         "type":"string",                         "maxLength":<i>n</i>                     }                 </pre> <p>This is a string.</p> <p>When <b>CHAR-OCCURS</b> =ARRAY :</p> <p><b>At mapping level 4.0 and lower</b></p> <pre data-bbox="863 569 1459 926">                     "field-name":{                         "maxItems":<i>m</i>,                         "minItems":<i>n</i>,                         "items":{                             "type":"object",                             "properties":{                                 "field-name":{                                     "type":"string",                                     "maxLength":1                                 }                             },                             "required":["field-                     name"]                         }                     }                 </pre> <p>This is an array of single characters.</p> <p><b>At mapping level 4.1 and higher</b></p> <p><b>TRUNCATE-NULL-ARRAYS</b> = DISABLED</p> <p>Both structured and primitive arrays map as follows.</p> <pre data-bbox="863 1178 1459 1283">                     "type":"array"                     "maxItems":<i>n</i>                     "minItems":<i>n</i>                     "items":{<i>data description JSON</i>}                 </pre> <p><b>TRUNCATE-NULL-ARRAYS</b> = ENABLED</p> <p>Primitive arrays map as above, and structured arrays as follows.</p> <pre data-bbox="863 1430 1459 1535">                     "type":"array"                     "maxItems":<i>n</i>                     "minItems":0                     "items":{<i>data description JSON</i>}                 </pre>

Table 8. Mapping reference for COBOL data description elements (continued)

COBOL data description	JSON schema definition
<p>PIC X OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDING ON <i>t</i></p> <p>PIC A OCCURS <i>n</i> TO <i>m</i>TIMES DEPENDING ON <i>t</i></p> <p>PIC G DISPLAY-1 OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDING ON <i>t</i></p> <p>PIC N OCCURS <i>n</i> TO <i>m</i>TIMES DEPENDING ON <i>t</i></p>	<p>When <b>CHAR-OCCURS</b> =STRING :</p> <pre data-bbox="836 304 1218 430"> "field-name":{   "type":"string",   "maxLength":<i>m</i>   "minLength":<i>n</i> } </pre>
<p>PIC N(<i>n</i>) USAGE NATIONAL When <b>CHAR-USAGE</b> =NATIONAL : PIC N( <i>n</i> )</p>	<p>At mapping level 4.0 and higher:</p> <pre data-bbox="836 861 1039 913"> "type":"string", "maxLength":<i>n</i> </pre> <p>At run time, CICS populates the application data structure field with UTF-16 data.</p>

## JSON schema to COBOL mapping

The DFHJS2LS utility program supports mappings between JSON schema and COBOL data structures.

### How JSON schema element names are converted to COBOL

The CICS assistants generate unique, valid field names for COBOL variables from the schema element names by using the following rules:

1. COBOL reserved words are prefixed with ' X '.  
For example, DISPLAY becomes XDISPLAY.
2. Characters other than A-Z, a-z, 0-9, or hyphen are replaced with ' X '.  
For example, monthly\_total becomes monthlyXtotal.
3. If the last character is a hyphen, it is replaced with ' X '.  
For example, ca-request- becomes ca-requestX.
4. 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.  
Should the above behavior be undesirable the user can specify MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING as input to the utility which disables the addition of one or two numeric digits to the second and subsequent instances of the name.
5. A JSON schema specifies that a variable has varying cardinality if it has a "type" value of "array" , and the keywords "minItems" and "maxItems" are omitted or have different values. If the schema specifies that the variable has varying cardinality, then field names are created with suffixes of "\_cont" and "\_num".

For more information, see “Variable arrays of elements in DFHJS2LS” on page 296.

6. A JSON schema specifies that a variable is optional if it does not appear in the "required" keyword array that is associated with the enclosing JSON schema "object" type. For optional fields, an additional field is generated with a suffix of \_num added to the element name. At run time this is zero to indicate the value was absent from the JSON data, and non-zero if the value was present in the JSON data.
7. Field names are limited to 28 characters. If a generated name, including the prefix and suffix, exceeds this length, the element name is truncated. If this name is not distinct, a unique numeric index will be added to the end.

## How JSON schema types are mapped to COBOL

DFHJS2LS map schema types to COBOL data description elements by using the specified mapping level according to [Table 9 on page 265](#). 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:

```
"textString": {
  "type": "string",
  "maxLength": 10000,
  "minLength": 1
}
```

maps to:

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

JSON Schema keyword	COBOL data description
All of: "type": "null" "type": []	Not supported
"\$schema": "http://json-schema.org/draft-04/schema#"	This keyword is ignored, but it is assumed to be compatible with the draft 04 JSON Schema specification.
"title": "same text" "description": "more text"	These keywords are ignored.
"format": "<predefined values>"	The "format" keyword is used to modify either the generated structure or runtime value. See the information later in this table for the supported use of "format".

Table 9. Mapping reference for JSON schema types (continued)

JSON Schema keyword	COBOL data description
<pre>"type": "array", "items": {&lt;JSON Sub-schema&gt;}, "additionalItems": false, "maxItems": m, "minItems": n</pre>	<p>Multidimensional arrays are supported from mapping level 4.3 or higher, and mixed type arrays are not supported.</p> <p>"additionalItems" is assumed to be false, and no other value is supported.</p> <p>If both "minItems" and "maxItems" are present, and they are equal, then the array is treated as fixed cardinality, otherwise it is treated as varying cardinality. See <a href="#">"Variable arrays of elements in DFHJS2LS"</a> on page 296.</p>
<pre>"type": "array", "uniqueItems": true</pre>	<p>"uniqueItems" is not supported with JSON arrays.</p>
<pre>"type": "object", "additionalProperties": false, "properties": { [ "&lt;element name&gt;": {&lt;JSON Sub-schema&gt; [ , ] ]* } } "required": [ [ "&lt;element name&gt;" [ , ] ]* ]</pre>	<p>The only form of JSON object that is currently supported is a fixed set of named elements.</p> <p>This generates a structure (or sub-structure) that uses the element names.</p> <p>"additionalProperties" is assumed to be false if a value is not set in the <b>ADDITIONAL-PROPERTIES-DEFAULT</b> parameter.</p> <p>Any element in the "properties" object is considered "optional" if it is not in the "required" array or if no "required" array exists. An "optional" element is given a variable ordinality of zero to X; where X is either 1 or the maximum number of items in the array, where that item is defined as an array. See <a href="#">"Variable arrays of elements in DFHJS2LS"</a> on page 296.</p>

Table 9. Mapping reference for JSON schema types (continued)

JSON Schema keyword	COBOL data description
<pre>"type": "object", "additionalProperties": true</pre>	<p>The object is mapped as in the previous example, with additional fields that support additional properties. The additionalProperties property is assumed to be false if not set in the <b>ADDITIONAL - PROPERTIES - DEFAULT</b> parameter.</p> <p>If enabled, space is allocated up to the value specified in the <b>ADDITIONAL - PROPERTIES - MAX</b> parameter. The number of characters in each space is set by the <b>ADDITIONAL - PROPERTIES - SIZE</b> parameter. Each individual property is mapped to a PIC X(z) field where z is defined by the <b>ADDITIONAL - PROPERTIES - SIZE</b> parameter. If the value of <b>ADDITIONAL - PROPERTIES - MAX</b> is greater than 0 then the properties are mapped in an equivalent fashion to that of an array with <b>maxItems</b> set.</p> <p><b>Note:</b> There are several ways to configure JSON support in CICS, including use of z/OS Connect for CICS. If you are using the older CICS Java Pipeline technology, Additional Properties are only supported if the com.ibm.cics.json.enableAxis2Handlers JVM system property is not set to true.</p>
<pre>"type": "object", "maxProperties": m, "minProperties": n, "patternProperties": {}, "dependencies":</pre>	<p>None of these keywords are supported with JSON objects.</p>
<pre>"type": "string" "maxLength": m "pattern": "&lt;regular expression&gt;", "minLength": l</pre>	<p>PIC X( z )</p> <p>where the value of z is based on m , but dependent on the settings of the <i>CHAR-VARYING</i> parameter.</p> <p>m is based on the "maxLength" keyword and treated as fixed length string.</p> <p>"pattern" and "minLength" restrictions are passed through to the language structure only as a comment.</p>
<pre>"type": "string" "maxLength": m</pre>	<p>When CCSID=1200 at mapping level 4.0 and higher: PIC N( z ) USAGE NATIONAL</p> <p>where the value of z is based on m , but dependent on the settings of the <b>CHAR-VARYING</b> parameter.</p> <p>m is based on the "maxLength" keyword and treated as fixed length string.</p>
<pre>"*name*": {   "type": "string",   "format": "date-time" }</pre>	<p>PIC S9(15) COMP-3</p> <p>All supported when DATETIME=PACKED15</p> <p>Note that "maxLength" and "minLength" are not supported for this format</p>

Table 9. Mapping reference for JSON schema types (continued)

JSON Schema keyword	COBOL data description
<pre>"*name*":{   "type":"string",   "format":"uri" }</pre>	<p>PIC X( <i>m</i> )</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string.</p> <p>When <b>CCSID=1200</b> at mapping level 4.0 and higher:            PIC N(<i>m</i>) USAGE NATIONAL</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string.</p>
<pre>"*name*":{   "type":"string",   "format":"base64Binary" }</pre>	<p>PIC X( <i>m</i> )</p> <p>where <i>m</i> is based on the "maxLength" keyword</p>
<pre>"*name*":{   "type":"string",   "format":"hexBinary" }</pre>	<p>PIC X( <i>m</i> )</p> <p>where <i>m</i> is based on the "maxLength" keyword</p>
<pre>"*name*":{   "type":"string",   "format":"&lt;predefined&gt;" }</pre>	<p>PIC X( <i>m</i> )</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string, and where &lt;predefined&gt; is one of: <i>email</i>, <i>hostname</i>, <i>ipv4</i>, or <i>ipv6</i>. A relevant "pattern" is used and passed to the comment.</p> <p>When <b>CCSID=1200</b> at mapping level 4.0 and higher:            PIC N( <i>m</i> ) USAGE NATIONAL</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string, and where &lt;predefined&gt; is one of: <i>email</i>, <i>hostname</i>, <i>ipv4</i>, or <i>ipv6</i>. A relevant "pattern" is used and passed to the comment.</p>
<pre>"type":"boolean"</pre>	<p>PIC X DISPLAY</p> <p>The value x'00' implies false, x'01' implies true.</p>
<pre>"type":"integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n</pre>	<p>"exclusiveMaximum" and "exclusiveMinimum" restrictions are passed to the language structure only as a comment.</p> <p>"multipleOf" is ignored.</p>

Table 9. Mapping reference for JSON schema types (continued)

JSON Schema keyword	COBOL data description
<pre>"type"="integer", minimum=0, maximum=255</pre>	<p>Mapping level 3.0 and below: PIC X DISPLAY</p> <p>Mapping level 4.0 and above (or when the INTEGER-AS-PIC9 parameter has been specified, regardless of the mapping level) :</p> <p>PIC 9( z ) COMP-5 SYNC</p> <p>or</p> <p>PIC 9( z ) DISPLAY</p> <p>where <math>10^{(z-1)} &lt; m \leq 10^z</math></p>
<pre>"type":"integer", minimum:-128, maximum:127</pre>	<p>Mapping level 3.0 and below: PIC X DISPLAY</p> <p>Mapping level 4.0 and above (or when the INTEGER-AS-PIC9 parameter has been specified, regardless of the mapping level) :</p> <p>PIC S9( z ) COMP-5 SYNC</p> <p>or</p> <p>PIC S9( z ) DISPLAY</p> <p>where <math>10^{(z-1)} &lt; m \leq 10^z</math></p>
<pre>"type":"integer", minimum:0, maximum;m</pre>	<p>PIC 9( z ) COMP-5 SYNC</p> <p>or</p> <p>PIC 9( z ) DISPLAY</p> <p>where <math>10^{(z-1)} &lt; m \leq 10^z</math></p>
<pre>"type":"integer", minimum:-m, maximum:m-1</pre>	<p>PIC S9( z ) COMP-5 SYNC</p> <p>or</p> <p>PIC S9( z ) DISPLAY</p> <p>where <math>10^{(z-1)} &lt; m \leq 10^z</math></p>
<pre>"type": "number", "maximum": m, "minimum": n, "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n</pre>	<p>"maximum", "minimum", "exclusiveMaximum" and "exclusiveMinimum" restrictions are passed to the language structure only as a comment.</p> <p>"multipleOf" is ignored.</p>
<pre>"type":"number" "format":"decimal"</pre>	<p>PIC 9( p )V9( n ) COMP-3</p> <p>where <i>p</i> and <i>n</i> are default values.</p>

Table 9. Mapping reference for JSON schema types (continued)

JSON Schema keyword	COBOL data description
<pre>"type": "number" "format": "float"</pre>	<p>Mapping level 1.1 and lower:</p> <ul style="list-style-type: none"> <li>• PIC X(32)</li> </ul> <p>Mapping level 1.2 and higher:</p> <ul style="list-style-type: none"> <li>• COMP-1</li> </ul> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values might not convert exactly from one representation to the other.</p> <p>Some extremely large or small values might not be valid for float data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-1 data types with fixed precision alternatives.</p>
<pre>"type": "number" "format": "double"</pre>	<p>Mapping level 1.1 and lower:</p> <ul style="list-style-type: none"> <li>• PIC X(32)</li> </ul> <p>Mapping level 1.2 and higher:</p> <ul style="list-style-type: none"> <li>• COMP-2</li> </ul> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values might not convert exactly from one representation to the other.</p> <p>Some extremely large or small values might not be valid for double data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-2 data types with fixed precision alternatives.</p>

Table 9. Mapping reference for JSON schema types (continued)

JSON Schema keyword	COBOL data description
<pre> oneOf: [ { "properties": { "A": { ... } } } ,         { "properties": { "B": { ... } } } ] anyOf: [ { "properties": { "A": { ... } } } } ,         { "properties": { "B": { ... } } } ] allOf: [ { "properties": { "A": { ... } } } } ,         { "properties": { "B": { ... } } } ] </pre>	<p>The logical paths through the array of options are merged together as though a single Object had been defined of type { "properties": { "A": ..., "B": ... } }. Each of the JSON properties is treated as optional. If the sub-options contain conflicting definitions for the same property name, an error message is issued; for example, if a single property is defined as both a String (in path A) and an Object (in path B), then such a definition is not supported and will result in an error message.</p> <p>It is possible for a JSON schema to define complex logical structures; however, the subtleties implied in the complex logical structures might be lost in the mapping to a language structure. The transformation process does not attempt to enforce the combinatorial rules from the schema; it only interacts with the language structure fields that indicate whether a given JSON property is present or absent.</p> <p>If the sub-options contain compatible definitions for the same property name, DFHJS2LS attempts to merge the associated pattern of constraints, though some subtleties might be lost in the process. For example, assume the following definition:</p> <pre> "A": { "oneOf": [ { "type": "string",                   "maxLength": 5}, {"type": "string",                   "minLength": 7, "maxLength": 8} ] } </pre> <p>"A" is defined as either a string up to 5 characters long or a string between 7 and 8 characters long. The merged composition might result in mappings to a string between 0 and 8 characters long, without recognizing that length of 6 characters is invalid.</p> <p>Most scenarios involving logical composition will map to simple language structures, but complicated logical composition might result in compromises during the mapping process. For best results, avoid using logical composition in a JSON schema to define alternative declarations for the same JSON property.</p>

**Note:**

- CICS cannot transform integer values greater than the maximum value for a signed long (2<sup>63</sup> - 1) unless they are enclosed within quotes.
- Minimum and maximum values specified in the schema for numeric types are used only to map to a COBOL datatype. Data is not validated against these values at run time.

Some of the schema types that are shown in [Table 9 on page 265](#) map to a COBOL format of COMP-5 SYNC or of DISPLAY, depending on the values (if any) that are specified in the minimum and maximum keywords:

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

```
"maximum": a
"minimum": -a
```

where *a* is a string of '9's.

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

```
"maximum": a
"minimum": 0
```

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 JSON schema mapping

The DFHLS2JS utility program supports mappings between C and C++ data types and JSON schema definitions.

### How C and C++ names are converted to JSON

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

1. Characters that are not valid in JSON property names are replaced with ' X '.  
For example, monthlY-tota1 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.

### How C and C++ data types are mapped to JSON

DFHLS2JS maps C and C++ data types to schema elements according to [Table 10 on page 273](#).

The `_Packed` qualifier is supported for structures.

#### Restriction:

- C and C++ types that are not shown in [Table 10 on page 273](#) are not supported by DFHLS2JS.
- Header files must contain a top level `struct` instance.
- You cannot declare a structure type that contains itself as a member.
- The following C and C++ data types are not supported:

```
decimal
long double
wchar_t (C++ only)
```

- The following are ignored if they are present in the header file.

#### Storage class specifiers:

```
auto
register
static
extern
mutable
```

#### Qualifiers

```
const
volatile
_Export (C++ only)
```

## Function specifiers

`inline` (C++ only)  
`virtual` (C++ only)

- 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 `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
<code>char[ z ]</code>	<pre>"type": "string" "maxLength": z</pre>
<code>char16_t[ n ]</code>	At mapping level 4.0 and higher: <pre>"type": "string" "maxLength": n</pre> At run time, CICS populates the application data structure field with UTF-16 data.
<code>char[8]</code> Supported at mapping level 3.0 and higher when DATETIME=PACKED15	<pre>"type": "string" "format": "date-time"</pre> The format of the timestamp is defined by <a href="#">RFC3339</a> .

Table 10. Mapping reference for C and C++ data types (continued)

C and C++ data type	Schema simpleType
char short int long long long	<pre data-bbox="613 254 841 323">"type": "integer", "minimum": - (n+1), "maximum": n</pre> <p data-bbox="597 352 1458 384">where <math>n</math> is the maximum value that can be represented by the primitive.</p>
unsigned char unsigned short unsigned int unsigned long unsigned long long	<pre data-bbox="613 491 829 560">"type": "integer", "minimum": 0, "maximum": n</pre> <p data-bbox="597 590 1458 621">where <math>n</math> is the maximum value that can be represented by the primitive.</p>
bool  (C++ only)	<pre data-bbox="613 726 813 751">"type": "boolean"</pre>
float  Supported at mapping level 1.2 and higher.	<pre data-bbox="613 848 818 896">"type": "number", "format": "float"</pre> <p data-bbox="597 926 1451 1146"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of float data types with fixed precision alternatives.</p>
double  Supported at mapping level 1.2 and higher.	<pre data-bbox="613 1194 829 1243">"type": "number", "format": "double"</pre> <p data-bbox="597 1272 1451 1493"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for double data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of double data types with fixed precision alternatives.</p>

Table 10. Mapping reference for C and C++ data types (continued)

C and C++ data type	Schema simpleType
<p><i>type name</i> [ <i>n</i> ]</p>	<p>For primitives:</p> <pre data-bbox="613 289 1057 478"> "array": {   "type": "array"   "maxItems": <i>n</i>   "minItems": <i>n</i>   "items": {     "type": "object",     "properties": {<i>name</i>: {<i>type</i> JSON}}     "required": [<i>name</i>]   } } </pre> <p>For structs:</p> <pre data-bbox="613 562 857 657"> "array": {   "type": "array"   "maxItems": <i>n</i>   "minItems": <i>n</i>   "items": {<i>type</i> JSON} } </pre> <p>Where <i>type JSON</i> is the JSON schema representation of the C or C++ type.</p>

## JSON schema to C and C++ mapping

The DFHJS2LS utility programs support mappings between the JSON schemas and C and C++ data types.

### How JSON schema element names are converted to C and C++

The CICS assistants generate unique, valid field 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. Duplicate names in the same scope are made unique by the addition of one or two numeric digits to the second and subsequent instances of the name.  
For example, three instances of `year` become `year`, `year1`, and `year2`.
4. A JSON schema specifies that a variable has varying cardinality if it has a "type" value of "array", and the keywords "minItems" and "maxItems" are omitted or have different values. If the schema specifies that the variable has varying cardinality, then field names are created with suffixes of "\_cont" and "\_num".  
For more information, see ["Variable arrays of elements in DFHJS2LS"](#) on page 296.
5. A JSON schema specifies that a variable is optional if it does not appear in the "required" keyword array that is associated with the enclosing JSON schema "object" type. For optional fields, an additional field is generated with a suffix of `_num` added to the element name. At run time this is zero to indicate the value was absent from the JSON data, and non-zero if the value was present in the JSON data.
6. Field names are limited to 50 characters. If a generated name, including any prefix and suffix, exceeds this length, the element name is truncated.

### How JSON schema type values are mapped to C and C++

DFHJS2LS maps JSON schema type values to C and C++ data types according to [Table 11 on page 276](#). 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.

<i>Table 11. Mapping reference for JSON schema type values</i>	
<b>JSON Schema keyword</b>	<b>C and C++ data type</b>
All of: <pre>"type": "null" "type": []</pre>	Not supported
<pre>"\$schema": "http://json-schema.org/draft-04/ schema#" </pre>	This keyword is ignored, but it is assumed to be compatible with the draft 04 JSON Schema specification.
<pre>"title": "same text" "description": "more text" </pre>	These keywords are ignored.
<pre>"format": "&lt;predefined values&gt;" </pre>	The "format" keyword is used to modify either the generated structure or runtime value. See the following information for the supported use of "format".
<pre>"type": "array", "items": {&lt;JSON Sub-schema&gt;}, "additionalItems": false, "maxItems": m, "minItems": n </pre>	<p>Multidimensional arrays are supported from mapping level 4.3 or higher, and mixed type arrays are not supported.</p> <p>"additionalItems" is assumed to be false, and no other value is supported.</p> <p>If both "minItems" and "maxItems" are present, and they are equal, then the array is treated as fixed cardinality, otherwise it is treated as varying cardinality. See <a href="#">“Variable arrays of elements in DFHJS2LS”</a> on page 296.</p>
<pre>"type": "array", "uniqueItems": true </pre>	"uniqueItems" is not supported with JSON arrays. The <JSON Sub-schema> must define a supported "type", but that "type" cannot be "array". This is a restriction on the language structure generated.

Table 11. Mapping reference for JSON schema type values (continued)

JSON Schema keyword	C and C++ data type
<pre>"type": "object", "additionalProperties": false, "properties": { [ "&lt;element name&gt;": {&lt;JSON Sub- schema&gt; } [, ] ]* } "required": [ [ "&lt;element name&gt;" [, ] ]* ]</pre>	<p>The only form of JSON object currently supported is a fixed set of named elements.</p> <p>This will generate a structure (or sub-structure) using the element names.</p> <p>"additionalProperties" is assumed to be false if a value is not set in the <b>ADDITIONAL - PROPERTIES - DEFAULT</b> parameter.</p> <p>Any element in the "properties" object is considered "optional" if it is not in the "required" array or if no "required" array exists. An "optional" element is given a variable ordinality of zero to X; where X is either 1 or the maximum number of items in the array, where that item is defined as an array. See <a href="#">“Variable arrays of elements in DFHJS2LS”</a> on page 296.</p>
<pre>"type": "object", "additionalProperties": true</pre>	<p>The object is mapped as in the previous example, with additional fields that support additional properties. The additionalProperties property is assumed to be false if not set in the <b>ADDITIONAL - PROPERTIES - DEFAULT</b> parameter.</p> <p>If enabled, space is allocated up to the value specified in the <b>ADDITIONAL - PROPERTIES - MAX</b> parameter. The number of characters in each space is set by the <b>ADDITIONAL - PROPERTIES - SIZE</b> parameter. Each individual property is mapped to a PIC X(z) field where z is defined by the <b>ADDITIONAL - PROPERTIES - SIZE</b> parameter. If the value of <b>ADDITIONAL - PROPERTIES - MAX</b> is greater than 0 then the properties are mapped in an equivalent fashion to that of an array with <b>maxItems</b> set.</p> <p><b>Note:</b> There are several ways to configure JSON support in CICS, including use of z/OS Connect for CICS. If you are using the older CICS Java Pipeline technology, Additional Properties are only supported if the <code>com.ibm.cics.json.enableAxis2Handlers</code> JVM system property is not set to true.</p>
<pre>"type": "object", "maxProperties": m, "minProperties": n, "patternProperties": {}, "dependencies":</pre>	<p>None of these keywords are supported with JSON objects.</p>

Table 11. Mapping reference for JSON schema type values (continued)

JSON Schema keyword	C and C++ data type
<pre>"type": "string" "maxLength": m "pattern": "regular expression", "minLength": l</pre>	<p>char[ z ]</p> <p>where the value of z is based on m , but dependent on the settings of the <b>CHAR-VARYING</b> parameter.</p> <p>m is based on the "maxLength" keyword and treated as fixed length string.</p> <p>"pattern" and "minLength" restrictions are passed through to the language structure only as a comment.</p>
<pre>"type": "string" "maxLength": m</pre>	<p>When CCSID=1200 at mapping level 4.0 and higher:</p> <p>char16_t[ z ]</p> <p>where the value of z is based on m , but dependent on the settings of the <b>CHAR-VARYING</b> parameter.</p> <p>m is based on the "maxLength" keyword and treated as fixed length string.</p>
<pre>"*name*": {   "type": "string",   "format": "date-time" }</pre>	<p>char[ 8 ]</p> <p>All supported when DATETIME=PACKED15</p>
<pre>"*name*": {   "type": "string",   "format": "uri" }</pre>	<p>char[ m ]</p> <p>where m is based on the "maxLength" keyword and treated as fixed length string.</p> <p>When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <p>char16_t[ m ]</p> <p>where m is based on the "maxLength" keyword and treated as fixed length string.</p>
<pre>"*name*": {   "type": "string",   "format": "base64Binary" }</pre>	<p>char[ m ]</p> <p>where m is based on the "maxLength" keyword.</p>
<pre>"*name*": {   "type": "string",   "format": "hexBinary" }</pre>	<p>char[ m ]</p> <p>where m is based on the "maxLength" keyword.</p>

Table 11. Mapping reference for JSON schema type values (continued)

JSON Schema keyword	C and C++ data type
<pre>"*name*":{   "type":"string",   "format":"&lt;predefined&gt;" }</pre>	<p>char[ <i>m</i> ]</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string, and where &lt;predefined&gt; is one of: <i>email</i>, <i>hostname</i>, <i>ipv4</i> or <i>ipv6</i>. A relevant "pattern" is used and passed to the comment.</p> <p>When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <p>char16_t[ <i>m</i> ]</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string, and where &lt;predefined&gt; is one of: <i>email</i>, <i>hostname</i>, <i>ipv4</i>, or <i>ipv6</i>. A relevant "pattern" is used and passed to the comment.</p>
<pre>"type":"boolean"</pre>	<p>bool (C++ only)</p> <p>short (C only)</p>
<pre>"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n</pre>	<p>"exclusiveMaximum" and "exclusiveMinimum" restrictions are passed to the language structure only as a comment.</p> <p>"multipleOf" is ignored.</p>
<pre>"type":"integer", minimum:-128, maximum:127</pre>	<p>signed char</p>
<pre>"type":"integer", minimum:0, maximum:255</pre>	<p>unsigned char</p>
<pre>"type":"integer", minimum:-32768, maximum:32767</pre>	<p>short</p>
<pre>"type":"integer", minimum:0, maximum:65535</pre>	<p>unsigned short</p>
<pre>"type":"integer", minimum:-2147483648, maximum:2147483647</pre>	<p>int</p>
<pre>"type":"integer", minimum:0, maximum:4294967295</pre>	<p>unsigned int</p>

Table 11. Mapping reference for JSON schema type values (continued)

JSON Schema keyword	C and C++ data type
<pre>"type": "integer", minimum: -9223372036854775808, maximum: 9223372036854775807</pre>	long long
<pre>"type": "integer", minimum: 0, maximum: 18446744073709551615</pre>	unsigned long long
<pre>"type": "number", "maximum": m, "minimum": n, "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n</pre>	<p>"maximum", "minimum", "exclusiveMaximum" and "exclusiveMinimum" restrictions are passed to the language structure only as a comment.</p> <p>"multipleOf" is ignored.</p>
<pre>"type": "number" "format": "float"</pre>	<p>Mapping level 1.1 and lower:</p> <ul style="list-style-type: none"> <li>• char[32]</li> </ul> <p>Mapping level 1.2 and higher:</p> <ul style="list-style-type: none"> <li>• float(*)</li> </ul> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of float data types with fixed precision alternatives.</p>
<pre>"type": "number" "format": "double"</pre>	<p>Mapping level 1.0 and lower:</p> <ul style="list-style-type: none"> <li>• char[32]</li> </ul> <p>Mapping level 1.2 and higher:</p> <ul style="list-style-type: none"> <li>• double(*)</li> </ul> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for double data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of double data types with fixed precision alternatives.</p>

Table 11. Mapping reference for JSON schema type values (continued)

JSON Schema keyword	C and C++ data type
<pre> oneOf: [ { "properties": { "A": { ... } } } ,         { "properties": { "B": { ... } } } ] anyOf: [ { "properties": { "A": { ... } } } } ,         { "properties": { "B": { ... } } } ] allOf: [ { "properties": { "A": { ... } } } } ,         { "properties": { "B": { ... } } } ] </pre>	<p>The logical paths through the array of options are merged together as though a single Object had been defined of type { "properties": { "A": ..., "B": ... } }. Each of the JSON properties is treated as optional. If the sub-options contain conflicting definitions for the same property name, an error message is issued; for example, if a single property is defined as both a String (in path A) and an Object (in path B), then such a definition is not supported and will result in an error message.</p> <p>It is possible for a JSON schema to define complex logical structures; however, the subtleties implied in the complex logical structures might be lost in the mapping to a language structure. The transformation process does not attempt to enforce the combinatorial rules from the schema; it only interacts with the language structure fields that indicate whether a given JSON property is present or absent.</p> <p>If the sub-options contain compatible definitions for the same property name, DFHJS2LS attempts to merge the associated pattern of constraints, though some subtleties might be lost in the process. For example, assume the following definition:</p> <pre> "A": { "oneOf": [ { "type": "string",                   "maxLength": 5}, {"type": "string",                   "minLength": 7, "maxLength": 8} ] } </pre> <p>"A" is defined as either a string up to 5 characters long or a string between 7 and 8 characters long. The merged composition might result in mappings to a string between 0 and 8 characters long, without recognizing that length of 6 characters is invalid.</p> <p>Most scenarios involving logical composition will map to simple language structures, but complicated logical composition might result in compromises during the mapping process. For best results, avoid using logical composition in a JSON schema to define alternative declarations for the same JSON property.</p>

**Note:**

- CICS cannot transform integer values greater than the maximum value for a signed long (2<sup>63</sup> - 1) unless they are enclosed within quotes.
- Minimum and maximum values specified in the schema for numeric types are used only to map to a C or C++ datatype. Data is not validated against these values at run time.

## PL/I to JSON schema mapping

The DFHLS2JS utility program supports mappings between PL/I data structures and JSON schema definitions. Because the Enterprise PL/I compiler and older PL/I compilers differ, two language options are supported: PLI-ENTERPRISE and PLI-OTHER.

### How PL/I names are converted to JSON

PL/I names are converted to JSON names according to the following rules:

1. Characters that are not valid in JSON property 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`.

### How PL/I data types are mapped to JSON

DFHLS2JS maps PL/I data types to schema elements according to [Table 12 on page 283](#).

#### Restriction:

- PL/I types that are not shown in [Table 12 on page 283](#) are not supported by DFHLS2JS.
- 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 that are specified as DECIMAL( $p, q$ ) are supported only when  $p \geq q$
- Data items that are 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 `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 `string` 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, leading and trailing white space will be removed and multiple spaces are replaced with a single space.
- DFHLS2JS does not fully implement the padding algorithms of PL/I; therefore, you must declare padding bytes explicitly in your data structure. DFHLS2JS issues a message if it detects that padding bytes are missing. Each top-level structure must start on a doubleword boundary and each byte in the structure must be mapped to the correct boundary.

For example, 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 fullword boundary.
- FIELD3 is 8 bytes long and must be aligned on a doubleword 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 is aligned at a fullword boundary, the compiler inserts three padding bytes immediately before FIELD1.

Because DFHLS2JS does not insert equivalent padding bytes, you must declare them explicitly before the structure is processed by DFHLS2JS. 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 more information about PL/I structural memory alignment requirements, see [Enterprise PL/I for z/OS product information](#).

Table 12. Mapping reference for PL/I data types	
PL/I data description	JSON schema definition
FIXED BINARY ( <i>n</i> )	<pre>"type": "integer", "minimum": - ( <i>n</i> + 1 ), "maximum": <i>n</i></pre> <p>where <i>n</i> is the maximum value that can be represented by the primitive.</p>
UNSIGNED FIXED BINARY( <i>n</i> ) <b>Restriction:</b> Enterprise PL/I only	<pre>"type": "integer", "minimum": 0, "maximum": <i>n</i></pre> <p>where <i>n</i> is the maximum value that can be represented by the primitive.</p>
FIXED DECIMAL( <i>n</i> , <i>m</i> )	<pre>"type": "number", "format": "decimal", "multipleOf": <i>x</i>, "maximum": <i>y</i>, "minimum": -<i>z</i></pre> <p>where:</p> <p><i>x</i> is the smallest unit available = <math>1 / 10^m</math>.</p> <p><i>y</i> is the maximum value that can be represented by the combination of <i>n</i> and <i>m</i>.</p> <p><i>z</i> is the maximum value that can be represented by the combination of <i>n</i> and <i>m</i>.</p>
FIXED DECIMAL (15) Supported at mapping level 3.0 and higher when <b>DATETIME=PACKED15</b>	<pre>"type": "string", "format": "date-time"</pre> <p>The format of the time stamp is defined by <a href="#">RFC3339</a>.</p>
BIT( <i>n</i> ) where <i>n</i> is a multiple of 8. Other values are not supported.	<pre>"type": "string" "maxLength": <i>m</i></pre> <p>where <math>m = n / 8</math></p>

Table 12. Mapping reference for PL/I data types (continued)

PL/I data description	JSON schema definition
<p>CHARACTER( <i>n</i> )</p> <p>VARYING and VARYINGZ are also supported at mapping level 1.2 and higher.</p> <p><b>Restriction:</b> VARYINGZ is supported only by Enterprise PL/I</p>	<pre>"type": "string" "maxLength": <i>n</i></pre>
<p>GRAPHIC( <i>n</i> )</p> <p>VARYING and VARYINGZ are also supported at mapping level 1.2 and higher.</p> <p><b>Restriction:</b> VARYINGZ is supported only by Enterprise PL/I</p>	<p>At a mapping level of 1.0 and 1.1, where <math>m = 2 * n</math> :</p> <pre>"type": "string" "maxLength": <i>m</i></pre> <p>At a mapping level of 1.2 or higher:</p> <pre>"type": "string" "maxLength": <i>n</i></pre>
<p>WIDECHAR( <i>n</i> )</p> <p><b>Restriction:</b> Enterprise PL/I only</p>	<p>At a mapping level of 1.0 and 1.1, where <math>m = 2 * n</math> :</p> <pre>"type": "string" "maxLength": <i>m</i></pre> <p>At a mapping level of 1.2 or higher:</p> <pre>"type": "string" "maxLength": <i>n</i></pre> <p>At mapping level 4.0 and higher, CICS populates the application data structure field with UTF-16 data.</p> <pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:maxLength value="<i>n</i>"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
<p>ORDINAL</p> <p><b>Restriction:</b> Enterprise PL/I only</p>	<pre>"type": "integer", "minimum": 0, "maximum": 255</pre>

Table 12. Mapping reference for PL/I data types (continued)

PL/I data description	JSON schema definition
<p>BINARY FLOAT ( <math>n</math> )                      where <math>n \leq 21</math></p> <p>Supported at mapping level 1.2 and higher.</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of BINARY FLOAT data types with fixed precision alternatives.</p>	<pre>"type": "number", "format": "float"</pre>
<p>BINARY FLOAT ( <math>n</math> )                      where <math>21 &lt; n \leq 53</math></p> <p>Values greater than 53 are not supported.</p> <p>Supported at mapping level 1.2 and higher.</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of BINARY FLOAT data types with fixed precision alternatives.</p>	<pre>"type": "number", "format": "double"</pre>

Table 12. Mapping reference for PL/I data types (continued)

PL/I data description	JSON schema definition
<p>DECIMAL FLOAT( <i>n</i> )                      where <math>n \leq 6</math></p> <p>Supported at mapping level 1.2 and higher.</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>	<pre>"type": "number", "format": "float"</pre>
<p>DECIMAL FLOAT( <i>n</i> )                      where <math>6 &lt; n \leq 16</math></p> <p>Values greater than 16 are not supported.</p> <p>Supported at mapping level 1.2 and higher.</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>	<pre>"type": "number", "format": "double"</pre>

Table 12. Mapping reference for PL/I data types (continued)

PL/I data description	JSON schema definition
<p><i>name ( n ) data description</i></p>	<p>For primitives:</p> <pre data-bbox="613 302 1187 485"> "array": {   "type": "array"   "maxItems": n   "minItems": n   "items": {     "type": "object",     "properties": {name: {data description JSON}}     "required": [name]   } } </pre> <p>For data declarations:</p> <pre data-bbox="613 575 1008 667"> "array": {   "type": "array"   "maxItems": n   "minItems": n   "items": {data description JSON} } </pre> <p>where <i>data description JSON</i> is the JSON schema representation of the PL/I data description.</p>

## JSON schema to PL/I mapping

The DFHJS2LS utility program supports mappings between JSON schemas 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.

### How JSON schema element names are converted to PL/I

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`.

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

2. 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`.

Should the above behavior be undesirable the user can specify **MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING** as input to the utility which disables the addition of one or two numeric digits to the second and subsequent instances of the name.

3. A JSON schema specifies that a variable has varying cardinality if it has a "type" value of "array", and the keywords "minItems" and "maxItems" are omitted or have different values. If the schema specifies that the variable has varying cardinality, then field names are created with suffixes of "\_cont" and "\_num".

For more information, see [“Variable arrays of elements in DFHJS2LS”](#) on page 296.

4. A JSON schema specifies that a variable is optional if it does not appear in the "required" keyword array that is associated with the enclosing JSON schema "object" type. For optional fields, an additional field is generated with a suffix of `_num` added to the element name. At run time this is zero to indicate the value was absent from the JSON data, and non-zero if the value was present in the JSON data.

- Field names are limited to 31 characters. If a generated name, including any prefix and suffix, exceeds this length, the element name is truncated.

The total length of the resulting name is 31 characters or less.

## How JSON schema types are mapped to PL/I

DFHJS2LS maps schema type values to PL/I data types according to [Table 13 on page 288](#). 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.

JSON Schema keyword	PL/I data description
All of: <pre>"type": "null" "type": []</pre>	Not supported
<pre>"\$schema": "http://json-schema.org/draft-04/ schema#" </pre>	This keyword is ignored, but it is assumed to be compatible with the draft 04 JSON Schema specification.
<pre>"title": "same text" "description": "more text" </pre>	These keywords are ignored.
<pre>"format": "&lt;predefined values&gt;" </pre>	The "format" keyword is used to modify either the generated structure or run-time value. See the following information for the supported use of "format".
<pre>"type": "array", "items": {&lt;JSON Sub-schema&gt;}, "additionalItems": false, "maxItems": m, "minItems": n </pre>	<p>Multidimensional arrays are supported from mapping level 4.3 or higher, and mixed type arrays are not supported.</p> <p>"additionalItems" is assumed to be false, and no other value is supported.</p> <p>If both "minItems" and "maxItems" are present, and they are equal, then the array is treated as fixed cardinality, otherwise it is treated as varying cardinality. See <a href="#">“Variable arrays of elements in DFHJS2LS” on page 296</a>.</p>
<pre>"type": "array", "uniqueItems": true </pre>	"uniqueItems" is not supported with JSON arrays. The <JSON Sub-schema> must define a supported "type", but that "type" cannot be "array". This is a restriction on the language structure generated.

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre>"type": "object", "additionalProperties": false, "properties": { [ "&lt;element name&gt;": {&lt;JSON Sub- schema&gt; } [, ] ]* } "required": [ [ "&lt;element name&gt;" [, ] ]* ]</pre>	<p>The only form of JSON object currently supported is a fixed set of named elements.</p> <p>This will generate a structure (or sub-structure) using the element names.</p> <p>"additionalProperties" is assumed to be false if a value is not set in the <b>ADDITIONAL - PROPERTIES - DEFAULT</b> parameter.</p> <p>Any element in the "properties" object is considered "optional" if it is not in the "required" array or if no "required" array exists. An "optional" element is given a variable ordinality of zero to X; where X is either 1 or the maximum number of items in the array, where that item is defined as an array. See <a href="#">"Variable arrays of elements in DFHJS2LS"</a> on page 296.</p>
<pre>"type": "object", "additionalProperties": true</pre>	<p>The object is mapped as in the previous example, with additional fields that support additional properties. The additionalProperties property is assumed to be false if not set in the <b>ADDITIONAL - PROPERTIES - DEFAULT</b> parameter.</p> <p>If enabled, space is allocated up to the value specified in the <b>ADDITIONAL - PROPERTIES - MAX</b> parameter. The number of characters in each space is set by the <b>ADDITIONAL - PROPERTIES - SIZE</b> parameter. Each individual property is mapped to a PIC X(z) field where z is defined by the <b>ADDITIONAL - PROPERTIES - SIZE</b> parameter. If the value of <b>ADDITIONAL - PROPERTIES - MAX</b> is greater than 0 then the properties are mapped in an equivalent fashion to that of an array with <b>maxItems</b> set.</p> <p><b>Note:</b> There are several ways to configure JSON support in CICS, including use of z/OS Connect for CICS. If you are using the older CICS Java Pipeline technology, Additional Properties are only supported if the <code>com.ibm.cics.json.enableAxis2Handlers</code> JVM system property is not set to true.</p>
<pre>"type": "object", "maxProperties": m, "minProperties": n, "patternProperties": {}, "dependencies":</pre>	<p>None of these keywords are supported with JSON objects.</p>

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre>"type": "string" "maxLength": m "pattern": "&lt;regular expression&gt;", "minLength": l</pre>	<p>char[ z ]</p> <p>where the value of z is based on m , but dependent on the settings of the <b>CHAR-VARYING</b> parameter.</p> <p>m is based on the "maxLength" keyword and treated as fixed length string.</p> <p>"pattern" and "minLength" restrictions are passed through to the language structure only as a comment.</p>
<pre>"type": "string" "maxLength": m</pre>	<p>When CCSID=1200 at mapping level 4.0 and higher:</p> <p>WIDECHAR( z )</p> <p>where the value of z is based on m , but dependent on the settings of the <b>CHAR-VARYING</b> parameter.</p> <p>m is based on the "maxLength" keyword and treated as fixed length string.</p>
<pre>"*name*": {   "type": "string",   "format": "date-time" }</pre>	<p>FIXED DECIMAL (15,0)</p> <p>All supported when DATETIME=PACKED15</p>
<pre>"*name*": {   "type": "string",   "format": "uri" }</pre>	<p>CHAR( m )</p> <p>where m is based on the "maxLength" keyword and treated as fixed length string.</p> <p>When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <p>WIDECHAR( m )</p> <p>where m is based on the "maxLength" keyword and treated as fixed length string.</p>
<pre>"*name*": {   "type": "string",   "format": "base64Binary" }</pre>	<p>CHAR( m )</p> <p>where m is based on the "maxLength" keyword.</p>
<pre>"*name*": {   "type": "string",   "format": "hexBinary" }</pre>	<p>CHAR( m )</p> <p>where m is based on the "maxLength" keyword.</p>

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre>"*name*":{   "type":"string",   "format":"&lt;predefined&gt;" }</pre>	<p>CHAR( <i>m</i> ) where <i>m</i> is based on the "maxLength" keyword and treated as a fixed length string, and &lt;predefined&gt; is one of: <i>email</i>, <i>hostname</i>, <i>ipv4</i> or <i>ipv6</i>. A relevant "pattern" is passed to the comment.</p> <p>When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <p>WIDECHAR( <i>m</i> )</p> <p>where <i>m</i> is based on the "maxLength" keyword and treated as fixed length string, and where &lt;predefined&gt; is one of: <i>email</i>, <i>hostname</i>, <i>ipv4</i>, or <i>ipv6</i>. A relevant "pattern" is used and passed to the comment.</p>
<pre>"type":"boolean"</pre>	<p>Mapping level 1.1 and lower:</p> <p><b>Enterprise PL/I</b> SIGNED FIXED BINARY (7)</p> <p><b>Other PL/I</b> FIXED BINARY (7)</p> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b> BIT(7) BIT(1)</p> <p><b>Other PL/I</b> BIT(7) BIT(1)</p> <p>where BIT(7) is provided for alignment and BIT(1) contains the Boolean mapped value.</p>
<pre>"type": "integer", "exclusiveMaximum": true, "exclusiveMinimum": true, "multipleOf": n</pre>	<p>"exclusiveMaximum" and "exclusiveMinimum" restrictions are passed to the language structure only as a comment.</p> <p>"multipleOf" is ignored.</p>
<pre>"type":"integer", minimum:-128, maximum:127</pre>	<p><b>Enterprise PL/I</b> SIGNED FIXED BINARY (7)</p> <p><b>Other PL/I</b> FIXED BINARY (7)</p>
<pre>"type":"integer", minimum:0, maximum:255</pre>	<p><b>Enterprise PL/I</b> UNSIGNED FIXED BINARY (8)</p> <p><b>Other PL/I</b> FIXED BINARY (8)</p>
<pre>"type":"integer", minimum:-32768, maximum:32767</pre>	<p><b>Enterprise PL/I</b> SIGNED FIXED BINARY (15)</p> <p><b>Other PL/I</b> FIXED BINARY (15)</p>

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre>"type": "integer",   minimum: 0,   maximum: 65535</pre>	<p><b>Enterprise PL/I</b> UNSIGNED FIXED BINARY (16)</p> <p><b>Other PL/I</b> FIXED BINARY (16)</p>
<pre>"type": "integer",   minimum: -2147483648,   maximum: 2147483647</pre>	<p><b>Enterprise PL/I</b> SIGNED FIXED BINARY (31)</p> <p><b>Other PL/I</b> FIXED BINARY (31)</p>
<pre>"type": "integer",   minimum: 0,   maximum: 4294967295</pre>	<p>Mapping level 1.1 and lower: <b>Enterprise PL/I</b> UNSIGNED FIXED BINARY(32)</p> <p>Mapping level 1.2 and higher: <b>Enterprise PL/I</b> CHAR( y ) where y is a fixed length that is less than 16 MB.</p> <p>All mapping levels: <b>Other PL/I</b> BIT(64)</p>
<pre>"type": "integer",   minimum: -9223372036854775808,   maximum: 9223372036854775807</pre>	<p>Mapping level 1.1 and lower: <b>Enterprise PL/I</b> SIGNED FIXED BINARY(63)</p> <p><b>Note:</b> The LIMITS compiler directive might affect how the PL/I compiler interprets this field. CICS expects the field to be of the declared size, but the compiler might optimize the field into a smaller space, which can result in a mismatch. To avoid such problems, use the LIMITS(FIXEDBIN(63)) compile-time option.</p> <p>Mapping level 1.2 and higher: <b>Enterprise PL/I</b> CHAR( y ) where y is a fixed length that is less than 16 MB.</p> <p>All mapping levels: <b>Other PL/I</b> BIT(64)</p>

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre>"type": "integer",   minimum: 0,   maximum: 18446744073709551615</pre>	<p>Mapping level 1.1 and lower:</p> <p><b>Enterprise PL/I</b> UNSIGNED FIXED BINARY(64)</p> <p><b>Note:</b> The LIMITS compiler directive might affect how the PL/I compiler interprets this field. CICS expects the field to be of the declared size, but the compiler might optimize the field into a smaller space, which can result in a mismatch. To avoid such problems, use the LIMITS(FIXEDBIN(63)) compile-time option.</p> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b> CHAR( y )</p> <p>where y is a fixed length that is less than 16 MB.</p> <p><b>Other PL/I</b> BIT(64)</p>
<pre>"type": "number"   "description": "decimal"</pre>	<p>FIXED DECIMAL( n , m )</p>
<pre>"type": "number"   "description": "float"</pre>	<p>Mapping levels 1.0 and 1.1:</p> <ul style="list-style-type: none"> <li>CHAR(32)</li> </ul> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b> DECIMAL FLOAT(6) HEXADEC</p> <p><b>Other PL/I</b> DECIMAL FLOAT(6)</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre>"type": "number" "description": "double"</pre>	<p>Mapping levels 1.0 and 1.1:</p> <ul style="list-style-type: none"> <li>• CHAR(32)</li> </ul> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b>            DECIMAL FLOAT(16) HEXADEC</p> <p><b>Other PL/I</b>            DECIMAL FLOAT(16)</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation used for JSON. Some values may not convert exactly from one representation to the other. Some extremely large or small values might not be valid for float data types. Some values may lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>

Table 13. Mapping reference for JSON schema types (continued)

JSON Schema keyword	PL/I data description
<pre> oneOf: [ { "properties": { "A": {...} } },         { "properties": { "B": {...} } } ], anyOf: [ { "properties": { "A": {...} } },         { "properties": { "B": {...} } } ], allOf: [ { "properties": { "A": {...} } },         { "properties": { "B": {...} } } ] </pre>	<p>The logical paths through the array of options are merged together as though a single Object had been defined of type { "properties": { "A":..., "B":... } }. Each of the JSON properties is treated as optional. If the sub-options contain conflicting definitions for the same property name, an error message is issued; for example, if a single property is defined as both a String (in path A) and an Object (in path B), then such a definition is not supported and will result in an error message.</p> <p>It is possible for a JSON schema to define complex logical structures; however, the subtleties implied in the complex logical structures might be lost in the mapping to a language structure. The transformation process does not attempt to enforce the combinatorial rules from the schema; it only interacts with the language structure fields that indicate whether a given JSON property is present or absent.</p> <p>If the sub-options contain compatible definitions for the same property name, DFHJS2LS attempts to merge the associated pattern of constraints, though some subtleties might be lost in the process. For example, assume the following definition:</p> <pre> "A": { "oneOf": [ { "type": "string",                   "maxLength": 5}, { "type": "string",                   "minLength": 7, "maxLength": 8} ] } </pre> <p>"A" is defined as either a string up to 5 characters long or a string between 7 and 8 characters long. The merged composition might result in mappings to a string between 0 and 8 characters long, without recognizing that length of 6 characters is invalid.</p> <p>Most scenarios involving logical composition will map to simple language structures, but complicated logical composition might result in compromises during the mapping process. For best results, avoid using logical composition in a JSON schema to define alternative declarations for the same JSON property.</p>

**Note:**

- CICS cannot transform integer values greater than the maximum value for a signed long (2<sup>63</sup> - 1) unless they are enclosed within quotes.
- Minimum and maximum values specified in the schema for numeric types are used only to map to a PL/I datatype. Data is not validated against these values at run time.

## Variable arrays of elements in DFHJS2LS

JSON can contain arrays of varying numbers of elements. In general JSON 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 JSON data.

An array with a varying number of elements is represented in the JSON schema by using the `minItems` and `maxItems` keywords in the schema with "type" value of "array" :

- The `minItems` keyword specifies the minimum number of times that the element can occur. It can have a value of 0 or any positive integer. It defaults to the value 0.
- The `maxItems` keyword 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 `minItems` keyword.
- If the `maxItems` keyword is missing it means the array is unbounded.

An optional field can be denoted by a variable array of "maxItems" :1, as in the following example of an optional 8 byte string called "component" :

```
"properties":
  {
    "component":
      {
        "type": "array",
        "maxItems": 1,
        "items":
          {
            "type": "string",
            "maxLength": 8
          }
      }
  },
"required": ["component"]
```

The same effect can be produced by not including the field name in the "required" keyword value:

```
"properties":
  { "component":
    {
      "type": "string",
      "maxLength": 8
    }
  }
}
```

In general, JSON schemas that contain varying numbers of elements do not map efficiently into a single high-level language data structure. 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 JSON data to application data, it populates these structures with the application data and the application reads them.
- When CICS transforms the application data to JSON data, it reads the application data in the structures that have been populated by the application.

The following examples illustrate the format of these data structures. 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.

### Example 1. Fixed number of elements

This example illustrates an element that occurs exactly three times:

```
"properties":
  { "component":
    {
      "type": "array",
      "maxItems": 3,
      "minItems": 3,
      "items":
```

```

        {
          "type": "string",
          "maxLength": 8
        }
      },
      "required": ["component"]
    }
  }
}

```

In this example, the number of times that the element occurs is known in advance, so 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
```

## Example 2. Varying number of elements at mapping level 2 and below

This example illustrates a mandatory element that can occur from one to five times:

```

"properties":
  {"component":
    {
      "type": "array",
      "maxItems": 5,
      "minItems": 1,
      "items":
        {
          "type": "string",
          "maxLength": 8
        }
      }
    }
  },
  "required": ["component"]
}

```

The main data structure contains a declaration of two fields. When CICS transforms the JSON data to binary data, the first field `component - num` contains the number of times that the element appears in the JSON data, 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 DFHJS-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 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 `DFHJS-component` data structure.

If `minItems="0"`, or is missing, and `maxItems="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 `DFHJS-component` data structure.

**Note:** If the JSON data consists of a single recurring element, `DFHJS2LS` 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.

## Example 3. Variable 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 [“Example 2. Varying number of elements at mapping level 2 and below”](#) on page 297 , 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 `maxItems` is less than or equal to the value of **INLINE-MAXOCCURS-LIMIT** , inline mapping is used.
- If `maxItems` 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 [“Example 2. Varying number of elements at mapping level 2 and below”](#) on page 297, 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.

#### Example 4. Nested variable arrays

Complex JSON schemas can contain variable recurring elements, which in turn contain variable 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:

```
"properties":
  {"component1":
    {
      "type": "array",
      "maxItems": 5,
      "minItems": 1,
      "items":
        {
          "type": "object",
          "properties":
            {"component2":
              {
                "type": "string",
                "maxLength": 8
              }
            },
          "required": ["component2"]
        }
    }
  },
  "required": ["component1"]
```

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

A third-level structure contains these elements:

```
01 DFHJS-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 DFHJS-component1.

Each instance of component2-cont names a different container, each of which contains the data structure mapped by the third-level structure DFHJS-component2.

To illustrate this structure, consider the fragment of JSON data that matches the example:

```
{ "component1":
  [
    { "component2": "string1" },
    { "component2": "string2" },
  ]
}
```

"component1" occurs three times. The first two 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 DFHJS-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 DFHJS-DATA.
- The container named in component1-cont.
- Two containers named in the first two instances of component2-cont.

## Optional structures and the required keyword

Data structures are defined by the JSON Schema "type" of "object". The schemas relate field names to individual types using the object provided by the "properties" keyword. The requirement for these fields to be part of the JSON data described by the JSON Schema is controlled by the array provided by the "required" keyword. This array lists all the field names that must be present in the JSON data. Optional fields are therefore represented by their absence from this array, or as the array is not allowed to be empty, the absence of the "required" keyword. In this case, all fields are optional.

Optional fields are treated as a variable array of 0 or 1 elements. This adds an additional field with the suffix "-num" appended to the element name. If the total length is more than 28 characters, the element name is truncated. At run time this will be non-zero to indicate the value was present in the JSON data and zero if it was not.

This example shows two fields, one required called "required-structure" and the other one optional called "optional-structure":

```
{
  "type": "object",
  "properties":
```

```

    {"required-structure":
      {
        "type":"string",
        "maxLength": 8
      },
      "optional-structure":
      {
        "type":"string",
        "maxLength": 8
      }
    },
    "required":
    ["required-structure"]
  }

```

The generated COBOL structure shows both fields, but the second is preceded by "optional-structure-num" which is an integer count of the elements, with 0 representing none and 1 that it is present. The value is set to indicate whether the "optional-structure" contains valid data or not.

```

03 OutputData.
06 required-structure PIC X(8).
06 optional-structure-num PIC S9(9) COMP-5 SYNC.
06 optional-structure PIC X(8).

```

## Support for UTF-16 in application data

CICS web services support conversion of UTF-16 encoded application data into XML or JSON and also XML or JSON into UTF-16 encoded application data. Use UTF-16 when you need to store and process data in multiple languages.

CICS SOAP and JSON web services support conversion of UTF-16 encoded application data into XML or JSON and also XML or JSON into UTF-16 encoded application data. Unicode is a variable-width encoding scheme that enables systems to handle data efficiently.

UTF-16 is a variable width encoding for Unicode, where each character is represented by 2 or 4 bytes. CICS web services support CCSID 1200 for application data, which is UTF-16 BE (big endian) with IBM Private Use Area. This behavior is consistent with UTF-16 support in all supported languages.

UTF-16 is supported at mapping level 4.0 and upwards. You can customize how application data is converted by using mapping settings in the assistants. For more information about XML mapping levels, see [Mapping levels for the CICS assistants](#) . For more information about JSON mapping levels, see [Mapping levels for the CICS JSON assistants](#).

**Note:** UTF-16 requires more processing time and is less storage efficient than EBCDIC encodings. Furthermore, mixing encoding types incurs extra runtime processing.

## Mapping UTF-16 from XML or JSON schema to language structures

Support for UTF-16 depends on how you create the web service. Mapping XML or JSON schema to language structures, also known as top-down mapping, has the following characteristics. If UTF-16 is enabled, all text fields are mapped to UTF-16 fields, whereas numeric display data types in COBOL are mapped as EBCDIC. To use UTF-16, set the CCSID parameter of DFHJS2LS, DFHSC2LS, or DFHWS2LS to 1200.

For example, if the following XML schema fragment were present in the WSDL:

```

<xsd:element name="myString" nillable="false">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="20"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

```

The DFHWS2LS assistant generates the following field in a COBOL language structure:

```

myString PIC N(20) USAGE NATIONAL

```

The CHAR-MULTIPLIER parameter of the web services assistants can be used to specify the length of a field the assistants generate.

### CHAR-MULTIPLIER

When you use UTF-16, the only valid values for the **CHAR-MULTIPLIER** parameter are 2 or 4 , where 2 is the default value.

**CHAR-MULTIPLIER** = 2 , where the schema describes a string of maxLength  $x$  , generates PIC  $N(x)$  . Setting **CHAR-MULTIPLIER** = 2 does not preclude the use of surrogate pairs in a UTF-16 string, but impacts the number of characters that fit in the field.

**CHAR-MULTIPLIER** = 4 generates PIC  $N(2x)$  . If **CHAR-MULTIPLIER** = 4 , the value at run time is padded if the string includes characters that can be expressed in a single encoding unit.

## Mapping UTF-16 from language structures to XML or JSON schema

Mapping from a language structure to XML or JSON schema, also known as bottom-up mapping, is managed differently to top-down mapping. If a UTF-16 string is declared in the language structure, then the data is interpreted by CICS as UTF-16 encoded, otherwise, data is assumed to be in an EBCDIC encoding. The CCSID parameter for DFHLS2JS, DFHLS2SC, or DFHLS2WS indicates the encoding of any EBCDIC text within the application data; it must not be set to indicate UTF-16.

The data types that are interpreted as UTF-16 characters are as follows: PIC  $N(n)$  in COBOL, WIDECHAR( $n$ ) in PL/I, and `char16_t[ $n$ ]` in C and C++.

The CHAR-USAGE parameter of the web services assistants can be used to specify data types.

### CHAR-USAGE

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.

If you want to mix national data types that contain UTF-16 and DBCS data in the same copybook, you can use the USAGE NATIONAL or USAGE DISPLAY-1 qualifiers on individual fields.

**Note:** DFHLS2WS, DFHLS2SC, and DFHLS2JS do not support the COBOL GROUP USAGE NATIONAL clause.

## Generating mappings from language structure

---

To create JavaScript Object Notation (JSON) from application data or application data from JSON, you create the mappings to describe how CICS should transform the data and JSON at run time. You can start from any application data record; for example, you can start with a COMMAREA, VSAM file, temporary storage queue, or an IBM Db2 record.

### Before you begin

Before you create the mappings, you must make sure that these preconditions are met:

- You must have a language structure that describes the application record in a partitioned data set. The language structure can be written in any of the high-level languages that are supported by the CICS JSON assistant: COBOL, PL/I, C, and C++.
- You must configure the user ID under which DFHLS2JS runs to use z/OS UNIX.
- The user ID must have read permission to access the language structure and write permission to put the output in the appropriate directories on z/OS UNIX.
- You must allocate sufficient storage to the user ID for the ID to run Java . You can use any supported version of Java.

## About this task

Use the CICS JSON assistant to create the data mappings for the application record. The CICS JSON assistant creates CICS bundle and issues error messages about any unsupported items that it identifies in your language structure. The reference information for the CICS JSON assistant lists the restrictions that apply to each high-level language.

## Procedure

Run the DFHLS2JS batch job. DFHLS2JS has optional parameters that you select to meet your requirements, such as selecting a particular code page. Use the following parameters as a minimum:

- Specify the high-level language of your language structure in the **LANG** parameter.
- Specify the name and location of a bundle resource in the **BUNDLE** parameter.
- Specify the mapping level in the **MAPPING-LEVEL** parameter. Although you can use any mapping level, to get the most advanced mapping options use the latest mapping level.
- Specify the location and the code page of the language structures that describe the application record in the **PDSMEM** and **PDSCP** parameters.
- Specify the name and location of the .json JSON schema file in the **JSON-SCHEMA** parameter. If the file does not exist DFHLS2JS creates the JSON schema, but not the directory structure.
- Specify the name that is used for the JSONTRANSFRM bundle resource. This name is used by applications to identify the JSON mappings.

The batch job creates a bundle directory structure on z/OS UNIX. The bundle directory has a META-INF subdirectory that contains the bundle manifest. The batch job also creates a JSON schema and a JSON binding in the bundle, using the file names that you specified for the **JSONTRANSFRM** and **JSON-SCHEMA** parameters.

Install the BUNDLE resource that specifies this JSON binding. The JSONTRANSFRM bundle resource dynamically creates a JSONTRANSFRM resource, which defines the location of the JSON schema and binding file. You can use the CICS Explorer Bundle and Bundle Parts operational views to check the status of installed BUNDLE resources.

## Results

A CICS bundle that contains one JSON transformation is generated.

The following example shows DFHLS2JS with the minimum set of parameters specified.

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
// SET QT=''
//JCLLIB JCLLIB ORDER=FPHLQ.SDFHMOBI
//JAVAPROG EXEC DFHLS2JS,
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/jsbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSONTRANSFRM=example.jsbind
JSON-SCHEMA=/u/exampleapp/example.json
/*
```

## What to do next

Write an application program to transform the application data to JSON and vice versa. You can use the same mappings for both transformations.

## Generating mappings from a JSON schema

To create application data from JavaScript Object Notation (JSON) or JSON from application data, you create the mappings to describe how CICS should transform the data and JSON at run time. You can start

from a JSON schema. After the language structure and the mapping are generated, you can develop CICS application using the language structure, and transform JSON to application data and vice versa.

## Before you begin

Before you create the mappings, you must make sure that these preconditions are met:

- You must have the JSON schema that describes a JSON record.
- You must configure the user ID under which DFHJS2LS runs to use z/OS UNIX.
- The user ID must have read permission to access the JSON schema and write permission to put the output in the appropriate directories on z/OS UNIX.
- You must allocate sufficient storage to the user ID for the ID to run Java . You can use any supported version of Java.

## About this task

Use the CICS JSON assistant to create the data mappings for the application record. The CICS JSON assistant creates a CICS bundle and issues error messages about any unsupported items that it identifies in your language structure. The reference information for the CICS JSON assistant lists the restrictions that apply to each high-level language. For more information, see [“DFHJS2LS: JSON schema to high-level language conversion for linkable interface” on page 243.](#)

## Procedure

Run the DFHJS2LS batch job. DFHJS2LS has optional parameters that you select to meet your requirements, such as selecting a particular code page. Use the following parameters as a minimum:

- Specify the name and location of a bundle resource in the **BUNDLE** parameter.
- Specify the name and location of the JSON schema file in the **JSON-SCHEMA** parameter.
- Specify the mapping level in the **MAPPING-LEVEL** parameter. Although you can use any mapping level, to get the most advanced mapping options use the latest mapping level.
- Specify the high-level language for the generated language structure in the **LANG** parameter.
- Specify the location and the code page of the language structures that describe the application record in the **PDSMEM** and **PDSCP** parameters. DFHJS2LS creates the language structure, but not the directory structure.
- Specify the name that is used for the JSONTRANSFRM bundle resource in CICS . This name is used by applications to identify the JSON mappings.

The batch job creates a bundle directory structure on z/OS UNIX. The bundle directory has a META-INF subdirectory that contains the bundle manifest. The batch job also creates a JSON binding and copies the JSON schema into the bundle directory, using the file names that you specified for the **JSONTRANSFRM** and **JSON-SCHEMA** parameters. The batch job also creates the language structure at the location specified in the **PDSMEM** and **PDSLIB** parameters.

Install the BUNDLE resource that specifies this JSON binding. The JSONTRANSFRM bundle resource dynamically creates a JSONTRANSFRM resource, which defines the location of the JSON schema and binding file. This bundle resource is visible in CICS when you view the contents of the installed bundle using the CICS Explorer . It is not a normal CICS resource, and is not visible when using CEMT or the CICSplex SM WUI.

## Results

A CICS bundle that contains one JSON transformation is generated. A language structure is generated.

The following example shows DFHJS2LS with the minimum set of parameters specified.

```
//JS2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT=' '
//JCLLIB JCLLIB ORDER=FPHLQ.SDFHMOBI
//JAVAPROG EXEC DFHJS2LS,
//INPUT.SYSUT1 DD *
```

```
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test2
LOGFILE=/u/exampleapp/jsbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=/u/exampleapp
PDSMEM=CPYBK2
JSONTRANSFRM=example.jsbind
JSON-SCHEMA=/u/exampleapp/example.json
/*
```

## What to do next

Write an application program to transform the application data to JSON and vice versa. You can use the same mappings for both transformations.

## Transforming application data to JSON by linking to DFHJSON

The JSON transformer linkable interface DFHJSON is a CICS-supplied program that can be called to perform transformation between application data and JSON. Your application program can transform application data to JSON by linking to DFHJSON.

**Note:** **EXEC CICS TRANSFORM JSONTODATA** and **TRANSFORM DATATOJSON** commands supersede the linkable interface, DFHJSON, making it possible to convert JSON data to a language structure, or vice versa, in your application.

## Before you begin

You must have an enabled JSONTRANSFRM bundle resource that defines the JSON binding and JSON schema. If you intend to perform transformations by using Java, you must have an Axis2 JVM server already running.

## About this task

Create or update an application program to link to the CICS supplied program DFHJSON to do the transformation.

## Procedure

1. The application program must create a channel, for example *MyChannelName*, and put the following containers into the channel.
  - DFHJSON-DATA
  - DFHJSON-TRANSFRM
  - DFHJSON-JVMSERVER (optional)

For more information about these containers, see [JSON transformer linkable interface containers](#).

2. Use the **EXEC CICS LINK PROGRAM** command to transform the data to JSON:

```
EXEC CICS LINK PROGRAM('DFHJSON') CHANNEL('MyChannelName')
```

3. Get containers DFHJSON-ERROR and DFHJSON-ERRORMSG, and check if any errors occurred during the transformation.

If a problem occurs during the transformation, an error code is returned in the DFHJSON-ERROR container and a detailed message in the DFHJSON-ERRORMSG container. For more troubleshooting information, see [Troubleshooting problems with JSON requests](#). For information about the errors that are generated during validation or execution of DFHJSON, see [DFHJSON-ERROR container](#).

4. Get container DFHJSON-JSON and make use of the JSON in your application.
5. Install the application.

## Results

When the application runs the **LINK** command, CICS checks the JSONTRANSFRM bundle resource to find the mappings in the JSON binding and transforms the application binary data to JSON by using the containers on the channel. The JSON is placed in the DFHJSON-JSON container on return. The JSON conforms to the JSON schema that is defined in the JSONTRANSFRM bundle resource.

## What to do next

You can also use the same mappings to transform JSON to application data. For more information, see [“Transforming JSON to application data by linking to DFHJSON” on page 306.](#)

# Transforming application data to JSON by using the TRANSFORM DATATOJSON API command

---

You can use the **TRANSFORM DATATOJSON** API command in your application to transform application data to JSON.

## Before you begin

You must have an enabled JSONTRANSFRM resource that defines the JSON binding and JSON schema.

## About this task

The application must use a channel-based interface.

## Procedure

1. Create a channel and put into the channel an input container that contains the application data to be converted.

**Note:** This channel will also have an output container that contains the JSON output when the **TRANSFORM DATATOJSON** command completes. Do not create the output container before issuing **TRANSFORM DATATOJSON** because the container is created and populated as part of the command itself.

2. Use the **TRANSFORM DATATOJSON** command to transform the data to JSON.  
For example:

```
EXEC CICS TRANSFORM DATATOJSON CHANNEL(ChannelName)  
INCONTAINER(InpContainerName) OUTCONTAINER(OutContainerName)  
TRANSFORMER(BundleName)
```

## Results

When the application runs the **TRANSFORM DATATOJSON** command, CICS checks the JSONTRANSFRM bundle resource to find the mappings in the JSON binding and transforms the application binary data to JSON by using the containers on the channel. On return, the JSON is placed in the container that is specified in the **OUTCONTAINER** option of the **TRANSFORM DATATOJSON** command. If the option is omitted, DFHJSON-JSON is used by default. The JSON conforms to the JSON schema that is defined in the JSONTRANSFRM bundle resource.

# Transforming JSON to application data by linking to DFHJSON

---

The JSON transformer linkable interface DFHJSON is a CICS-supplied program that can be called to perform transformation between application data and JSON. Your application program can transform JSON to application data by linking to DFHJSON.

**Note:** **EXEC CICS TRANSFORM JSONTODATA** and **TRANSFORM DATATOJSON** commands supersede the linkable interface, DFHJSON, making it possible to convert JSON data to a language structure, or vice versa, in your application.

## Before you begin

You must have an enabled JSONTRANSFRM resource that defines the JSON binding and JSON schema. If you intend to perform transformations by using Java, you must have an Axis2 JVM server already running.

## About this task

Create or update an application program to link to the CICS supplied program DFHJSON to do the transformation.

## Procedure

1. Create a channel, for example *MyChannelName*, and put the following containers into the channel.
  - DFHJSON-JSON
  - DFHJSON-TRANSFRM
  - DFHJSON-JVMSEVR

For more information about these containers, see [JSON transformer linkable interface containers](#).

2. Use the **EXEC CICS LINK PROGRAM** API command to transform the data to JSON:

```
EXEC CICS LINK PROGRAM('DFHJSON') CHANNEL('MyChannelName')
```

3. Get containers DFHJSON-ERROR and DFHJSON-ERRORMSG, and check if any errors occurred during the transformation.

If a problem occurs during the transformation, an error code is returned in the DFHJSON-ERROR container and a detailed message in the DFHJSON-ERRORMSG container. For more troubleshooting information, see [Troubleshooting problems with JSON requests](#). For information about the errors that are generated during validation or execution of DFHJSON, see [DFHJSON-ERROR container](#).

4. Install the application program.

## Results

When the application runs the **LINK PROGRAM** command, CICS checks the JSONTRANSFRM resource to find the mappings in the JSON binding and transforms the JSON to the application data using the containers on the channel. The application data is placed in the DFHJSON-DATA bit container on return.

## What to do next

You can also use the same mappings to transform application data to JSON. For more information, see [“Transforming application data to JSON by linking to DFHJSON” on page 304](#).

# Transforming JSON to application data by using the TRANSFORM JSONTODATA API command

---

You can use the **TRANSFORM JSONTODATA** API command in your application to transform JSON to application data.

## Before you begin

You must have an enabled JSONTRANSFRM resource that defines the JSON binding and JSON schema.

## About this task

The application must use a channel-based interface.

## Procedure

1. Create a channel and put into the channel an input container that contains the JSON to be converted.

**Note:** This channel will also have an output container that contains the converted data when the **TRANSFORM JSONTODATA** command completes. Do not create the output container before issuing **TRANSFORM JSONTODATA** because the container is created and populated as part of the command itself.

2. Use the **TRANSFORM JSONTODATA** command to transform JSON to application data.  
For example:

```
EXEC CICS TRANSFORM JSONTODATA CHANNEL(ChannelName)  
INCONTAINER(InpContainerName) OUTCONTAINER(OutContainerName)  
TRANSFORMER(BundleName)
```

## Results

When the application runs the **TRANSFORM JSONTODATA** command, CICS checks the JSONTRANSFRM bundle resource to find the mappings in the JSON binding and transforms the JSON to the application binary data by using the containers on the channel. On return, the converted data is placed in the container that is specified in the **OUTCONTAINER** option of the **TRANSFORM JSONTODATA** command. If the option is omitted, DFHJSON-DATA is used by default.

## Handling invalid and uninitialized application-supplied data

---

How to tolerate invalid data that is found during transformation between JSON and application data.

Service enabling an existing application results in CICS becoming aware of the contents of the COMMAREA or containers that are used by that application. The WSBind file that is deployed to CICS contains information about the data format, including the names, location, and data type of each individual field. CICS uses this information to facilitate the transformation between the XML/JSON data representation, and the application data.

If the application data is not consistent with the information that is stored in the WSBind file, CICS reports a problem and fails to transform the data. For example, an application interface is changed but the WSBind file is not regenerated and redeployed. This type of problem often results in a DFHPI1010 error message.

A more subtle variant of this condition can occur in scenarios where the application has left fields that are deliberately uninitialized. For example, a signed packed decimal field might contain low values (null bytes). When CICS processes that field, it cannot determine that the invalid value is deliberately uninitialized. CICS displays the DFHPI1010 error message and the data transformation fails. Applications can leave fields uninitialized for many reasons. For example,

- A language structure might describe both the input and output data formats, but the uninitialized field was for input only.

- The application might detect an error, set a response code, and return without the other output fields initialized.
- Conditional logic within the application might control whether the field is considered to be meaningful.

The ideal response to this scenario is for the application to be changed to ensure that no invalid values are passed to CICS for processing. An alternative option is to set the Assistant's DATA-SCREENING option to DISABLED at the time that the WSBIND file is generated. This option causes CICS to tolerate bad application supplied values, and to replace such values with a default value, typically zero. These options can make service enablement of existing applications simpler, but makes error detection more difficult, and must be used with care. If data screening is disabled, it's more likely that application generated data errors is not detected by CICS.

A related condition can be experienced where application generated arrays are left partially unpopulated. For example, consider an array of 1000 records of data; the application might initialize the first 10 records. CICS generates an XML or JSON representation of the data and populates all 1000 fields, although 990 are empty. The ideal response to this problem is to introduce an OCCURS DEPENDING ON clause to the application to indicate precisely how many records are considered to be populated. An alternative is to use the TRUNCATE-NULL-ARRAYS option of the Assistants when the WSBIND file is generated. This option instructs CICS to attempt to detect uninitialized data, and to truncate the array at that point. Use of this option can result in elegant JSON/XML data that is generated from ambiguous and uninitialized application data, but introduces the risk of accidental data loss if the array content is indistinguishable from uninitialized data.

The best solution is to have applications produce unambiguous data that is entirely consistent with the language structure that describes that data. Use of the TRUNCATE-NULL-ARRAYS=ENABLED and DATA-SCREENING=DISABLED options can cause CICS to tolerate imperfect application-supplied data, but they introduce an element of risk and uncertainty in the process.

The following example demonstrates how to use these options.

## Example 1: toleration of decimal fields

Testing the automatic correction of invalid data in a decimal field.

### About this task

This scenario shows how the default value 0 can be automatically set in decimal fields that contain invalid data.

### Procedure

1. Generate the JSON schema and necessary artifacts for CICS to transform between JSON and COBOL application data.
  - a) Prepare the copybook.

```

03 BAD-DATA.
05 NORMAL-NUM PIC 9(2).
05 NORMAL-CHAR PIC X(3).
05 PZONED-DECIMAL PIC S9(4) DISPLAY.
05 NZONED-DECIMAL PIC S9(4) DISPLAY.
05 UZONED-DECIMAL PIC 9(4) DISPLAY.
05 NORMAL-CHAR2 PIC X(3).
05 PBINARY PIC S9(4) BINARY.
05 NBINARY PIC S9(4) COMP.
05 UBINARY PIC 9(4) COMP.
05 NORMAL-NUM2 PIC 9(3).
05 PPACKED-DECIMAL PIC S9(4) COMP-3.
05 NPACKED-DECIMAL PIC S9(4) COMP-3.
05 UPACKED-DECIMAL PIC 9(4) COMP-3.
05 NORMAL-NUM3 PIC 9(2).
05 NORMAL-CHAR3 PIC X(3).
05 FLOAT-ZONED PIC S9(4)V99.
05 FLOAT-PACKED PIC S9(4)V99 COMP-3.

```

- b) Submit the JCL.

Set DATA-SCREENING to DISABLED to activate the function.

```
//GENJSON JOB ('accounting information',name),CLASS=M,REGION=0M,  
// MSGCLASS=A,NOTIFY=&SYSUID  
//JCLLIB JCLLIB ORDER=ZZZZ.A.ZOSCONN.JCL  
//LS2JS EXEC DFHLS2JS,  
// JAVADIR='/java/java7_64/J7.0_64',  
// USSDIR='cics.ts.test/tolerate',  
// PATHPREF='',  
// TMPDIR='/tmp',  
// TMPFILE=''  
//INPUT.SYSUT1 DD *  
PDSLIB=ZZZZ.A.ZOSCONN.COPYBOOK  
REQMEM=BADDATA  
RESPMEM=BADDATA  
JSON-SCHEMA-REQUEST=/u/zzzz/json/ls2js/baddata1_request.json  
JSON-SCHEMA-RESPONSE=/u/zzzz/json/ls2js/baddata1_response.json  
LANG=COBOL  
LOGFILE=/u/zzzz/json/ls2js/baddata1.log  
MAPPING-LEVEL=4.1  
DATA-SCREENING=DISABLED  
CHAR-VARYING=COLLAPSE  
PGMNAME=baddata1  
URI=/baddata1  
PGMINT=COMMAREA  
WSBIND=/u/zzzz/json/ls2js/baddata1.wsbinding  
/*
```

c) Check that the generated JSON schemas (request and response) and WSBind files were created successfully.

2. Create the application program.

a) In COBOL program, assign a value to all fields except decimals.

```
MOVE LOW-VALUE TO BAD-DATA.  
MOVE 11 TO NORMAL-NUM.  
MOVE 'AAA' TO NORMAL-CHAR.  
MOVE 'BBB' TO NORMAL-CHAR2.  
MOVE 222 TO NORMAL-NUM2.  
MOVE 'CCC' TO NORMAL-CHAR3.
```

3. Define CICS resources.

a) Define and install COBOL program.

b) Define and install TCPIP SERVICE.

c) Based on your parser, define and install PIPELINE.

4. Test the application.

a) Run PIPELINE SCAN. For the webservice described in the WSbind file, use INSERVICE.

b) Send the JSON request.

```
{ "BADDATA10operation": { "bad_data": { "normal_num": 12 } } }
```

c) Check the response.

## Results

Default values are assigned to the uninitialized fields.

```
{  
  "BADDATA10operationResponse":  
    {  
      "bad_data":  
        {  
          "normal_num": 11,  
          "normal_char": "AAA",  
          "pzed_decimal": 0,  
          "nzoned_decima": 0,  
          "uzoned_decimal": 0,  
          "normal_char2": "BBB",  
          "pbinary": 0,  
          "nbinary": 0,  
        }  
      }  
    }  
}
```

```
        "ubinary": 0,  
        "normal_num2": 222,  
        "ppacked_decimal": 0,  
        "npacked_decimal": 0,  
        "upacked_decimal": 0,  
        "normal_num3": 0,  
        "normal_char3": "CCC",  
        "float_zoned": 0,  
        "float_packed": 0  
    }  
}
```

---

## Chapter 5. Developing with XML

For any purpose, you can write application programs to transform application binary data into XML and vice versa. CICS supports a number of high-level languages and provides an XML assistant to map how the data is transformed during runtime processing. CICS uses the same technology for mapping application data to XML in SOAP messages, as part of the web services support.

### Before you begin

You must have Java installed to run the XML assistant and to optionally perform validation when CICS transforms the data or XML.

### About this task

The advantage of using this approach to transform application data to and from XML is that CICS goes beyond the capabilities offered by an XML parser. CICS can interpret the XML and perform record-based conversions of the application data. Therefore, it is easier and faster for you to create applications that work with XML using this approach.

The CICS XML assistant is a supplied utility that helps you to create the required artifacts to transform application binary data to XML or transform XML to application binary data. The XML assistant can create the artifacts in a bundle directory or another specified location on z/OS UNIX.

### Procedure

1. Create the mappings using the XML assistant.
2. Create the XMLTRANSFORM resources in CICS to make the mappings available.
3. Create or update an application program to use the **TRANSFORM** API command.  
The application must use a channel-based interface.
4. Run the application to test that the transformation works as you intended.  
You can turn on validation to check that CICS converts the data correctly.  
These steps are explained in more detail in the following topics.

---

## The CICS XML assistant

The CICS XML assistant is a set of batch utilities that can help you to transform XML into high-level language structures and vice versa. The assistant supports rapid deployment of applications that perform XML processing with the minimum amount of programming effort.

Using the XML assistant for CICS reduces the amount of code that you must write to parse or construct XML; CICS transforms data between XML fragments and the data structure of an application program.

The XML assistant can create an XML schema from a simple language structure, or a language structure from an existing XML schema, and supports COBOL, C/C++, and PL/I. It also generates metadata that CICS uses at run time to automatically convert XML data to binary application data or vice versa; the metadata is defined in an XML binding and stored on z/OS UNIX. The schema for the XML binding is in `/usr/lpp/cicsts/cicsts52/schemas/xmltransform/` directory on z/OS UNIX.

The CICS XML assistant comprises two utility programs:

#### **DFHLS2SC**

This utility generates an XML schema and binding from a language structure.

#### **DFHSC2LS**

This utility generates an XML binding and language structure that you can use in your application programs. You can use either a WSDL document or an XML schema as input.

The JCL procedures to run both programs are in the *hlq* .XDFHINST library, where *hlq* is the high-level qualifier of your CICS installation.

The relevant usage mode for the DFHLS2SC or DFHSC2LS procedure depends on your requirements:

- “[DFHLS2SC: high-level language to XML schema conversion](#)” on page 312
- “[DFHSC2LS: XML schema to high-level language conversion](#)” on page 320

For information about the data mappings between high-level language structures and XML schemas or WSDL documents, see the following topics:

- “[How the CICS assistants map between high-level languages and XML schemas](#)” on page 329
- “[Mapping levels for the CICS assistants](#)” on page 332
- “[Data mapping limitations when using the CICS assistants](#)” on page 330
- “[Variable arrays of elements](#)” on page 368

## DFHLS2SC: high-level language to XML schema conversion

The DFHLS2SC cataloged procedure generates an XML schema and an XML binding file from a high-level language structure. Use DFHLS2SC when you want to create a CICS program that can parse or create XML.

### Job control statements for DFHLS2SC

#### **JOB**

Starts the job.

#### **EXEC**

Specifies the procedure name (DFHLS2SC).

#### **INPUT.SYSUT1 DD**

Specifies the input. The input parameters are usually specified in the input stream. However, you can define them in a data set or in a member of a partitioned data set.

### Symbolic parameters

The following symbolic parameters are defined in DFHLS2SC:

#### **JAVADIR = *path***

Specifies the name of the Java directory that is used by DFHLS2SC. The value of this parameter is appended to `/usr/lpp/` giving 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.

#### **PATHPREFIX = *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 **PATHPREFIX** 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 DFHLS2SC 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 DFHLS2SC uses to construct the names of the temporary workspace files.

The default value is `SC2WS`.

**PATHMAIN =path**

Specifies the main part of the name of the CICS TS directory in the UNIX system services file system. The default value is `/usr/lpp/cicsts`.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHMAIN** parameter.

**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 the value specified by the **PATHMAIN** parameter.

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**

DFHLS2SC 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/LS2SC.in  
/tmp/LS2SC.out  
/tmp/LS2SC.err
```

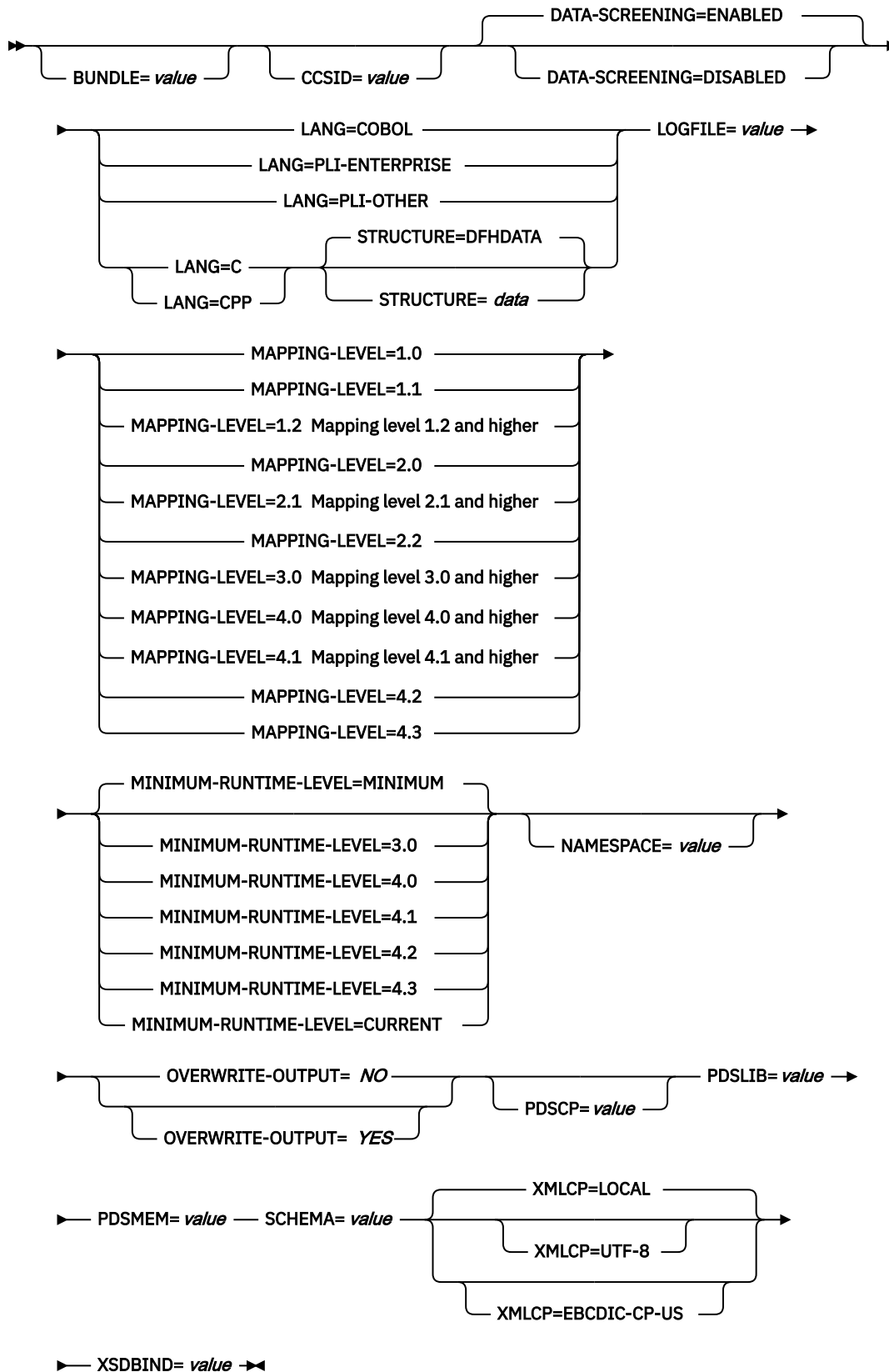
**Important:** DFHLS2SC does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHLS2SC run concurrently and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading 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.

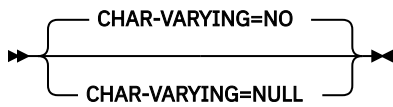
These temporary files are deleted before the end of the job.

**Important:** The utility will only run under the default USS shell. If you have configured a different shell (such as the bash shell) to launch, based on your `.profile`, then the utility will fail to run without issuing any error messages. If you use a different shell you must alter the configuration of your `.profile` to use the default USS shell before running the utility.

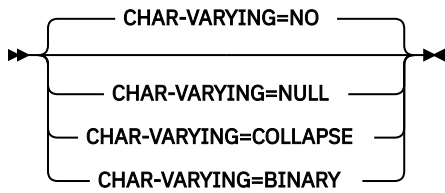
# Input parameters for DFHLS2SC



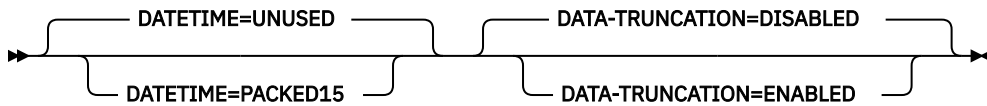
## Mapping level 1.2 and higher



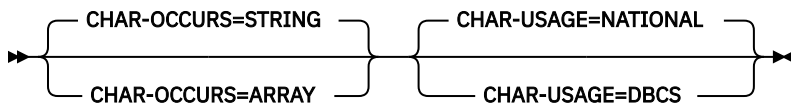
**Mapping level 2.1 and higher**



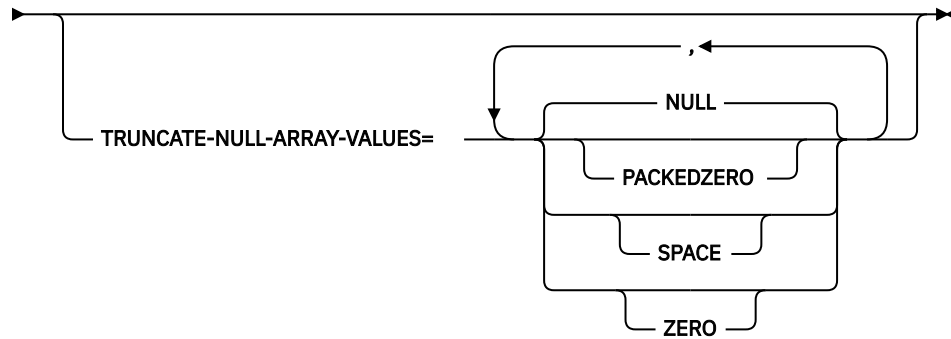
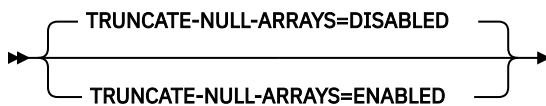
**Mapping level 3.0 and higher**



**Mapping level 4.0 and higher**



**Mapping level 4.1 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 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:

```
XSDBIND=/path/xsdbinddir*
                /app1
```

is equivalent to

```
XSDBIND=/path/xsdbinmdir/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

### **BUNDLE = value**

Specifies the path and name of the bundle directory on z/OS UNIX. If you specify this value, the XML assistant generates a bundle containing the XSD binding. The path information on this parameter overrides any path information on the **XSDBIND** parameter.

You can optionally specify an archive file rather than a directory name. The XML assistant supports .zip and .jar archives. However, you must uncompress the archive before trying to install the BUNDLE resource.

If you do not specify this parameter, CICS places the XML schema and binding in the location specified on the **XSDBIND** parameter.

### **CCSID = value**

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

You can use this parameter with any mapping level.

### **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. This parameter does not apply to Enterprise and Other PL/I language structures. 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.

#### **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 an XML schema. 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++.

#### **COLLAPSE**

Character fields are mapped to an `xsd:string`. Trailing and embedded white space in the field is not included in the XML schema; for example `<space>AB<space><space><space>C<space>` becomes `AB<space>C`. 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 as an `xsd:base64binary` data type 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 behavior 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 DBCS encoded data.

**NATIONAL**

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

**DATA-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

**ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

**DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

**DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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.

**LANG = COBOL | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

## CPP

C++

### **LOGFILE = *value***

The fully qualified z/OS UNIX name of the file into which DFHLS2SC writes its activity log and trace information. DFHLS2SC 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 DFHLS2SC, or the parsing or creation of XML in CICS.

### **MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

Specifies the level of mapping for the assistant to use when generating the XML binding and language structures. You are recommended to use the most recent mapping level that is available. If you are creating an XML binding for an Atom feed, you must use a mapping level of 3.0.

### **MINIMUM-RUNTIME-LEVEL = { MINIMUM | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | 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.

#### **3.0**

Specify runtime level 3.0 or above if you want to use the CICS XML assistant and take advantage of advanced data mappings.

#### **4.0**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, 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.

#### **4.1**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

#### **4.2**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

#### **4.3**

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

#### **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.

### **NAMESPACE = *value***

Specifies the namespace for CICS to use in the generated XML schema. For Atom feeds, CICS provides this namespace in the Atom feed together with the Atom namespace.

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

### **OVERWRITE-OUTPUT = **NO** | **YES****

Controls whether existing CICS BUNDLES on the file system can be overwritten.

#### **NO**

Any existing BUNDLE is not replaced. If an existing BUNDLE is found DFHLS2SC issues error message DFHPI9689E and terminates.

#### **YES**

Any existing BUNDLE is replaced. If an existing BUNDLE is found then message DFHPI9683W is issued to inform you that the file has been replaced.

**PDSCP = value**

Specifies the code page that is used in the partitioned data set members, where *value* is a CCSID number or a Java code page number. If you do not specify this parameter, 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 high-level language data structures to be processed.

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

**PDSMEM = value**

Specifies the name of the partitioned data set member that contains the high-level language structures to be processed.

**SCHEMA = value**

The fully qualified z/OS UNIX name of the file into which the XML schema is written. The XML schema conforms to the WSDL 2.0 specification. DFHLS2SC creates the file, but not the directory structure, if it does not already exist.

**STRUCTURE = { DFHDATA | data }**

The name of the top-level data structure in C and C++. The default is DFHDATA.

**TRUNCATE-NULL-ARRAYS = { DISABLED | ENABLED }**

Specifies how structured arrays are processed at mapping level 4.1 or higher. If enabled, CICS will attempt to recognize empty records within an array (see TRUNCATE-NULL-ARRAY-VALUES for more information about identifying empty records). If five consecutive empty array records are detected, the array is truncated at the first such record when generating XML/JSON. This truncation capability is only enabled for arrays with structured content, arrays of simple primitive fields are not subject to truncation. Truncation of arrays can result in a more concise representation of the data in JSON/XML, but is not without risk. If five consecutive data records are misidentified as uninitialized storage (perhaps because they legitimately contain low values), data loss can be experienced. If TRUNCATE-NULL-ARRAYS is enabled and TRUNCATE-NULL-ARRAY-VALUES is not set, then the default value for TRUNCATE-NULL-ARRAY-VALUES is used.

**TRUNCATE-NULL-ARRAY-VALUES = { NULL | PACKEDZERO | SPACE | ZERO }**

Specifies which values are treated as empty for TRUNCATE-NULL-ARRAYS processing at mapping level 4.1 or higher. By default, the null value (0x00, or low-values) is treated as empty. If all of the bytes of storage within a record of a structured array contain nulls, then the entire record is considered to be empty. One or more of the NULL, PACKEDZERO, SPACE and ZERO values can be specified in a comma separated list.

**NULL**

Implies a null character (0x00).

**PACKEDZERO**

Implies a positive signed packed decimal zero (0x0C), a negative signed packed decimal zero (0x0D), or an unsigned packed decimal zero (0x0F).

**SPACE**

Implies an SBCS EBCDIC space (0x40).

**ZERO**

Implies an unsigned zoned decimal zero (0xF0).

Any matching combination of the selected bytes within a structured array record will cause the entire record to be identified as empty.

If TRUNCATE-NULL-ARRAY-VALUES has a value defined, TRUNCATE-NULL-ARRAYS must be enabled.

**XSDBIND = value**

The fully qualified z/OS UNIX name of the XSD binding. DFHLS2SC creates the file, but not the directory structure, if it does not already exist. If the BUNDLE parameter is specified, exclude the path. The file extension is .xsdbind.

## DFHSC2LS: XML schema to high-level language conversion

The DFHSC2LS cataloged procedure generates a high-level language data structure and an XML binding from an XML schema or WSDL document. Use DFHSC2LS when you want to create a CICS program that can parse or create XML. This topic lists the job control statements, symbolic parameters, input parameters and their descriptions for DFHSC2LS.

### Job control statements for DFHSC2LS

#### **JOB**

Starts the job.

#### **EXEC**

Specifies the procedure name (DFHSC2LS).

#### **INPUT.SYSUT1 DD**

Specifies the input. The input parameters are usually specified in the input stream. However, you can define them in a data set or in a member of a partitioned data set.

### Symbolic parameters

The following symbolic parameters are defined in DFHSC2LS:

#### **JAVADIR = *path***

Specifies the name of the Java directory that is used by DFHSC2LS. The value of this parameter is appended to `/usr/lpp/` giving 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 **PATHPREF** parameter.

#### **TMPDIR = *tmpdir***

Specifies the location of a directory in z/OS UNIX that DFHSC2LS 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 DFHSC2LS uses to construct the names of the temporary workspace files.

The default value is `SC2LS`.

#### **PATHMAIN = *path***

Specifies the main part of the name of the CICS TS directory in the UNIX system services file system.

The default value is `/usr/lpp/cicsts`.

Typically, you do not specify this parameter. The default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **PATHMAIN** parameter.

#### **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 the value specified by the **PATHMAIN** parameter.

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

DFHSC2LS 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/SC2LS.in  
/tmp/SC2LS.out  
/tmp/SC2LS.err
```

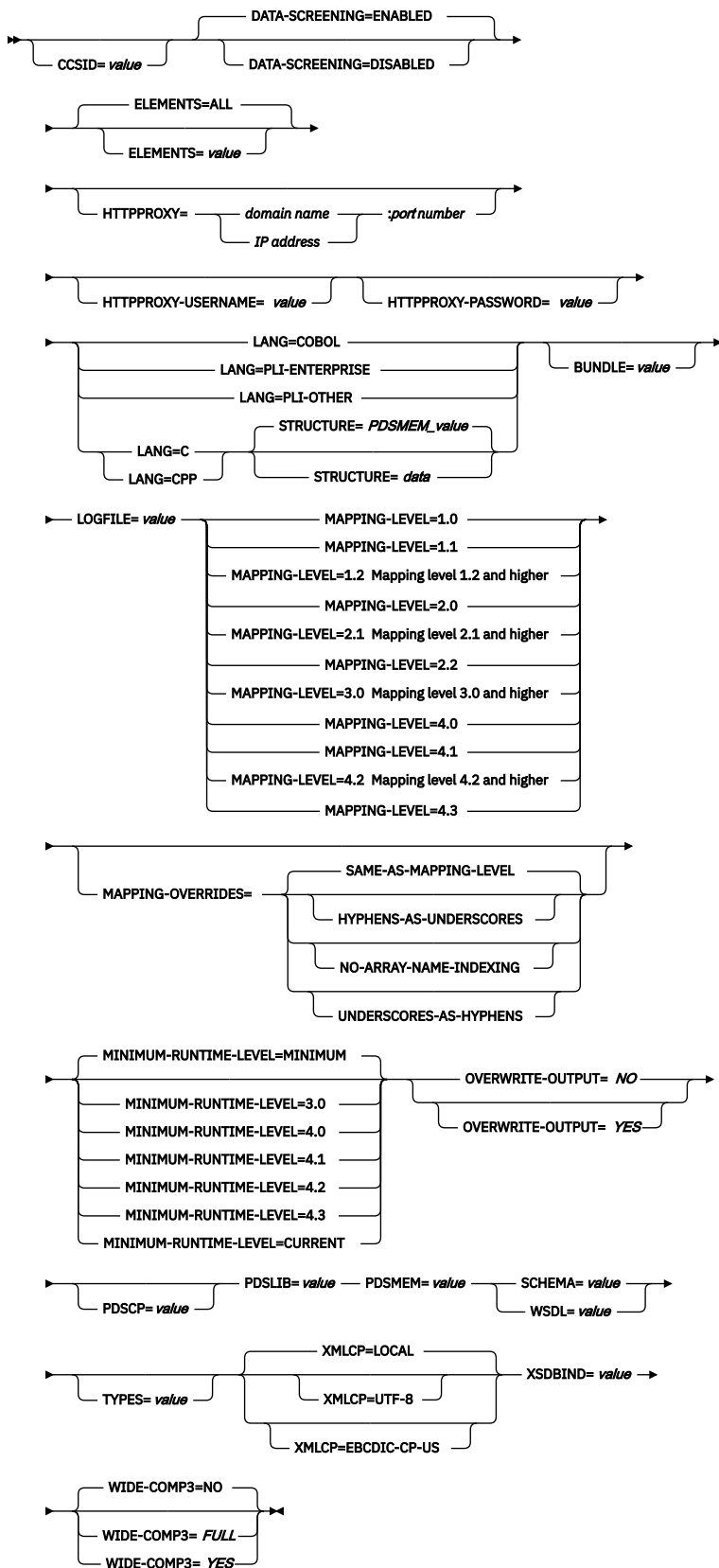
**Important:** DFHSC2LS does not lock access to the z/OS UNIX files or the data set members. Therefore, if two or more instances of DFHSC2LS run concurrently, and use the same temporary workspace files, nothing prevents one job from overwriting the workspace files while another job is using them, leading 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.

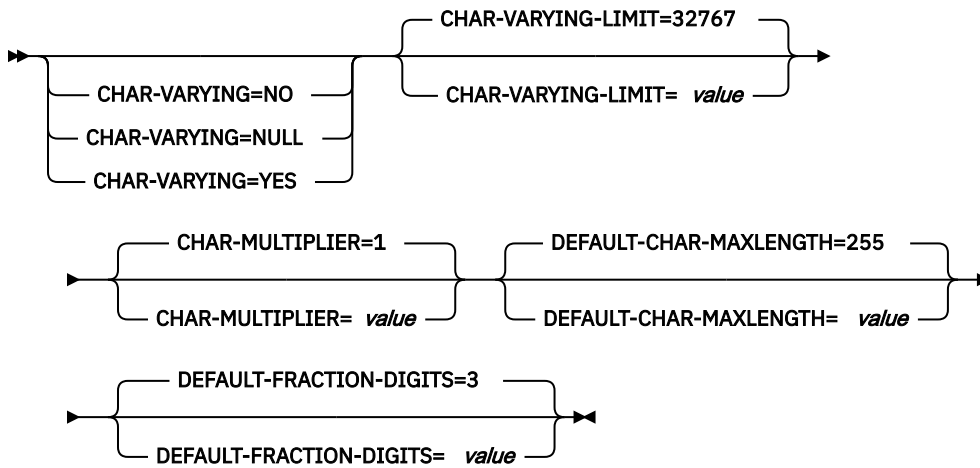
These temporary files are deleted before the end of the job.

**Important:** The utility will only run under the default USS shell. If you have configured a different shell (such as the bash shell) to launch, based on your .profile, then the utility will fail to run without issuing any error messages. If you use a different shell you must alter the configuration of your .profile to use the default USS shell before running the utility.

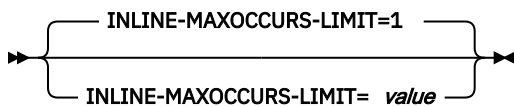
# Input parameters for DFHSC2LS



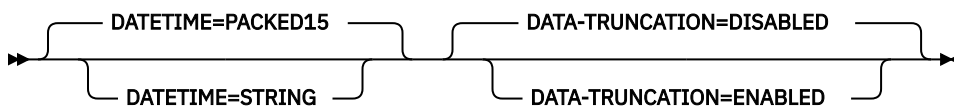
## Mapping level 1.2 and higher



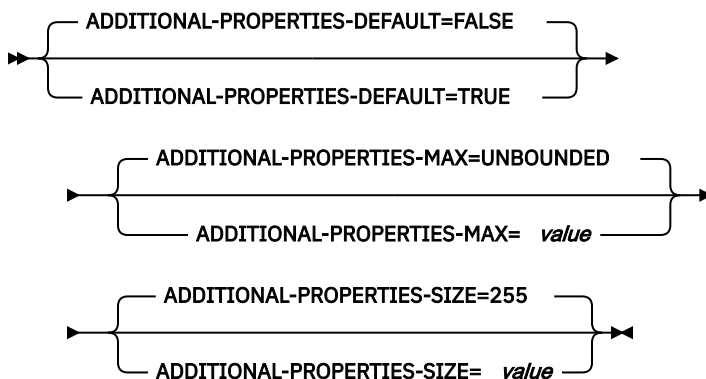
### Mapping level 2.1 and higher



### Mapping level 3.0 and higher



### Mapping level 4.2 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 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:

```
XSDBIND=xsbindir*
                /app1
```

is equivalent to

```
XSDBIND=xsdbinddir/app1
```

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

## Parameter descriptions

### **ADDITIONAL-PROPERTIES-DEFAULT = { true | false }**

Indicates whether JSON schema objects that do not explicitly declare support for Additional Properties are interpreted as supporting them or not. Additional JSON properties are any properties within a JSON object that are not pre-defined in the JSON Schema. These properties are typically rejected by the data transformation mechanism as unexpected extra data. If **ADDITIONAL-PROPERTIES-DEFAULT** is set to TRUE, or if the JSON schema explicitly sets `additionalProperties:true` for an object, then space is allocated in the generated copybooks to hold such values. Applications can interact with those values using the associated fields in the copybooks.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-MAX = { 0-20 | UNBOUNDED }**

Indicates how many Additional Properties are supported for a JSON object that supports them. See **ADDITIONAL-PROPERTIES-DEFAULT**. The generated copybooks will contain structures suitable for addressing any additional properties. By default, there is no maximum constraint placed on the number of properties that are supported. The copybooks are generated in a similar fashion to arrays with no constraints and use containers. This parameter can be used to apply a maximum constraint that can be used in combination with the **INLINE-MAXOCCURS-LIMIT** parameter to cause a fixed length array to be allocated for the maximum number of properties, thereby avoiding the need for containers.

You can use this parameter at a mapping level of 4.2 or higher.

### **ADDITIONAL-PROPERTIES-SIZE = { 16-32767 | 255 }**

Indicates the maximum size for each of the JSON additional properties. If a JSON object supports additional properties, as defined by **ADDITIONAL-PROPERTIES-DEFAULT**, then the generated copybooks will have bindings to support properties up to the number specified by **ADDITIONAL-PROPERTIES-MAX**. By default, the maximum value supported for each additional property is 255 characters. A field of that size will be generated into the copybooks that are produced. This size can be customized by setting the **ADDITIONAL-PROPERTIES-SIZE** parameter. For example, a JSON object is processed that is found to contain the following property:

```
"example": { "notes": "this extra property was not defined in the JSON Schema" }
```

If the copybooks have been generated to support additional properties, then that entire value will be passed to the application for processing. The value begins with the leading quotation mark before the property's key, and ends with the trailing right brace in the property's value. It's approximately 100 characters in this example. The value used for **ADDITIONAL-PROPERTIES-SIZE** must be large enough to hold the largest such value that might occur. If the allocated buffer is too small for the value that is processed, an error response is generated.

You can use this parameter at a mapping level of 4.2 or higher.

### **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.

### **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 consists of two related elements: the 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.

#### **DATA-SCREENING = { ENABLED | DISABLED }**

Specifies whether application supplied data is screened for errors.

##### **ENABLED**

Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error, and message DFHPI1010 is issued. An error response is returned to the application.

##### **DISABLED**

Values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID\_PACKED\_DEC and INVALID\_ZONED\_DEC error responses that are generated from uninitialized output fields.

#### **DATA-TRUNCATION = { DISABLED | ENABLED }**

Specifies whether 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 that xsd:dateTime fields are mapped to CICS ABSTIME data format or to text:

**PACKED15**

xsd:dateTime fields are mapped to CICS ABSTIME format.

**STRING**

xsd:dateTime fields are mapped to text. This mapping is the same as all previous mappings levels.

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

**DEFAULT-CHAR-MAXLENGTH = { 255 | value }**

Specifies the default array length of character data in characters for mappings where no length is implied in the XML schema document or WSDL 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 to 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 PL/I the valid range is 0-30.

**ELEMENTS = { ALL | value }**

Defines a list of global element local names to enable. The default value of ALL indicates that all global elements are enabled.

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

If your XML schema or WSDL document contains references to other XML schema or WSDL files that are located on the Internet, and the system on which you are running DFHSC2LS uses a proxy server to access the Internet, specify either 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-USERNAME = value**

Specifies the HTTP proxy user name to be used with **HTTPPROXY-PASSWORD** if the system on which you are running DFHSC2LS 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-PASSWORD = value**

Specifies the HTTP proxy password to be used with **HTTPPROXY-USERNAME** if the system on which you are running DFHSC2LS 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 inline variable repeating content is used based on the maxOccurs attribute of the XML attribute.

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 to 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 you have few instances of long length, container-based mapping is preferable; if you have many instances of short length, inline mapping is likely to be preferable.

**LANG = COBOL | PLI-ENTERPRISE | PLI-OTHER | C | CPP**

Specifies the programming language of the high-level language structure:

**COBOL**

COBOL

**PLI-ENTERPRISE**

Enterprise PL/I

**PLI-OTHER**

A level of PL/I other than Enterprise PL/I

**C**

C

**CPP**

C++

**LOGFILE = value**

The fully qualified z/OS UNIX name of the file into which DFHSC2LS writes its activity log and trace information. DFHSC2LS 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 DFHSC2LS.

**MAPPING-LEVEL = { 1.0 | 1.1 | 1.2 | 2.0 | 2.1 | 2.2 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 }**

Specifies the level of mapping for the assistant to use when generating the XML binding and language structures. You are recommended to use the most recent mapping level that is available; for DFHSC2LS, you are recommended to use a mapping level of 3.0 or higher.

**3.0**

The `xsd:dateTime` data type maps to the CICS ASKTIME format.

**4.0**

Use this mapping level with a region at CICS TS 5.3 or later when you want to use UTF-16.

**4.1**

For truncatable array support, use this mapping level with a region at CICS TS 5.3 or later.

**4.2**

For additional properties, use this mapping level with a region at CICS TS 5.4 or later.

**4.3**

For multidimensional array support, use this mapping level with a region at CICS TS 5.4 or later.

**MAPPING-OVERRIDES = { SAME-AS-MAPPING-LEVEL | [HYPHENS-AS-UNDERSCORES] | [NO-ARRAY-NAME-INDEXING] | [UNDERSCORES-AS-HYPHENS] }**

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

**SAME-AS-MAPPING-LEVEL**

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

**HYPHENS-AS-UNDERSCORES**

For PL/I only. This parameter converts any hyphens in the XML document to underscores rather than the character X, to improve the readability of the generated PL/I language structures. For more information, see [XML schema to PL/I mapping](#). This option is enabled automatically at mapping level 4.2.

**NO-ARRAY-NAME-INDEXING**

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

**UNDERSCORES-AS-HYPHENS**

For COBOL only. Converts any underscores in the XML document to hyphens instead of the character X. This option improves 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” on page 343](#).

This option is enabled automatically at Mapping level 4.0.

**MINIMUM-RUNTIME-LEVEL = { MINIMUM | 3.0 | 4.1 | 4.2 | 4.3 | CURRENT }**

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

## MINIMUM

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

### 3.0

Specify runtime level 3.0 or above if you want to use the CICS XML assistant and take advantage of advanced data mappings.

### 4.0

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, 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.

### 4.1

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.3, or later. With this runtime level, you can use a mapping level of 4.1 or earlier for the **MAPPING-LEVEL** parameter.

### 4.2

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.2 or earlier for the **MAPPING-LEVEL** parameter.

### 4.3

The generated web service binding file deploys successfully into a region at CICS TS for z/OS, Version 5.4, or later. With this runtime level, you can use a mapping level of 4.3 or earlier for the **MAPPING-LEVEL** parameter.

## CURRENT

Use this runtime level to deploy the generated XML binding into a CICS region that has the same runtime environment as the region used to generate the XML binding.

## OVERWRITE-OUTPUT = **NO** | **YES**

Controls whether existing CICS BUNDLES on the file system can be overwritten.

### NO

Any existing BUNDLE is not replaced. If an existing BUNDLE is found, DFHLS2SC issues error message DFHPI9689E and terminates.

### YES

Any existing BUNDLE is replaced. If an existing BUNDLE is found, DFHLS2SC issues message DFHPI9683W to inform you that the file has been replaced.

## PDSCP = *value*

Specifies the code page that is used in the partitioned data set members, where *value* is a CCSID number or a Java code page number. If you do not specify this parameter, 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.

## PDSMEM = *value*

Specifies the 1-6 character prefix that DFHSC2LS uses to generate the names of the partitioned data set members that will contain the high-level language structures.

DFHSC2LS generates a partitioned data set member for each operation. It generates the member name by appending a number to the prefix, with a maximum member name length of 8 characters.

## SCHEMA = *value*

The fully qualified z/OS UNIX name of the file from which the XML schema is read. DFHSC2LS creates the file, but not the directory structure, if it does not already exist.

You can use either an XML schema or a WSDL document as input to DFHSC2LS. You must specify either this parameter or the **WSDL** parameter, but not both, to indicate from where the input comes.

## STRUCTURE = { *PDSMEM\_value* | *data* }

The name of the top-level data structure in C and C++. The default value is the value of the **PDSMEM** parameter.

**TYPES = value**

Defines a list of global type local names to enable. A *value* of ALL indicates that all global types are enabled. By default, global types are not enabled.

**WIDE-COMP3 = { NO | FULL | YES }**

Controls the maximum size of the packed decimal variable length in the generated COBOL or PL/I language structure.

**FULL**

For COBOL and PL/I. DFHJS2LS generates a packed decimal field that is large enough to hold all valid values. The maximum size is 31 digits. This is the default.

**NO**

For COBOL only. 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**

For COBOL only. DFHJS2LS supports the maximum size of 31 when generating the COBOL language structure type COMP-3.

**Note:** The NO and YES options generate fields that are unable to represent all valid values; the FULL option avoids this problem. However, the FULL option does allow some invalid values to be represented in the packed decimal field. For example, if a schema indicates that there are a maximum of five digits and a maximum of two fractional digits, the FULL option will generate a packed decimal field that allows for seven digits, and this allows space for valid values such as 25000 and 999.99, but also provides space for some invalid values such as 9999.99. When you use the FULL option, take care not to generate invalid values in application data.

**WSDL = value**

The fully qualified z/OS UNIX name of the WSDL document.

You can use either an XML schema or a WSDL document as input to DFHSC2LS. You must specify either this parameter or the **SCHEMA** parameter, but not both, to indicate from where the input comes.

**XMLCP = { LOCAL | UTF-8 | EBCDIC-CP-US }**

Specifies the code page that is used to generate the XML binding.

**LOCAL**

This value is the default. It specifies that the XML is generated using the local code page and no encoding tag is generated in the XML schema.

**UTF-8**

Specifies that the XML is generated using the UTF-8 code page. An encoding tag is generated in the XML schema. If you specify this option, you must ensure that the encoding remains correct when copying the XML schema between different platforms.

**EBCDIC-CP-US**

Specifies that the XML is generated using the US EBCDIC code page. An encoding tag is generated in the XML schema.

**XSDBIND = value**

The fully qualified z/OS UNIX name of the XML binding. DFHSC2LS creates the file, but not the directory structure, if it does not already exist. The file extension is `.xsdbind`.

## How the CICS assistants map between high-level languages and XML schemas

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. The CICS assistants 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.

- 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.

**Stabilized feature:** Web Services Description Language (WSDL) 2.0 in CICS is stabilized. Use WSDL 1.1 as the de facto standard for SOAP-based web services. See also [Stabilization notices and discontinued functions](#).

## 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 378.

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:date` 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.

### Improved support at mapping level 4.0 and higher

When the mapping level is 4.0 or higher, the CICS assistants support the following mapping improvements:

At mapping level 4.0 and higher, DFHLS2SC and DFHLS2WS support the COBOL OCCURS DEPENDING ON clause and supports mapping of COBOL character arrays into XML strings. You can set this behavior by using the **CHAR-OCCURS** parameter on the CICS assistants.

- You must specify the parameter **DATA-TRUNCATION=ENABLED**.
- Complex OCCURS DEPENDING ON is not supported. This limitation means that OCCURS DEPENDING ON is only supported for the last field of a structure.
- CICS does not support qualified names (using the 'OF' keyword) as the target of an OCCURS DEPENDING ON clause, for example `FIELD1 OF STRUCTURE1`.
- CICS does not support the UNBOUNDED keyword. You must specify the maximum size of the table that is expected by the application.

**Note:** DFHLS2WS and DFHLS2SC do not support the COBOL `GROUP USAGE NATIONAL` clause.

At mapping level 4.0 and higher, CICS web services support the conversion of application data that is encoded using UTF-16 Unicode.

- When you use DFHLS2WS or DFHLS2SC, you can enable this behavior by using language-specific data types for UTF-16.
- When you use DFHWS2LS or DFHSC2LS, you can enable this behavior by setting `CCSID=1200`.
- CICS supports only a single Unicode code page, *UTF-16BE with IBM Private Use Area* (CCSID 1200).
- Conversion of application data that is encoded using UTF-8 is not supported.

**Note:** DFHLS2WS and DFHLS2SC do not support the COBOL `GROUP USAGE NATIONAL` clause.

### 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 must set the **MAPPING-LEVEL** parameter in the CICS assistants to the latest level.

Each level of mapping inherits the function of the previous mapping, where the highest level of mapping offers 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 4.3

Mapping level 4.3 is compatible with CICS TS V5.4 with APAR PI88519, and higher.

Mapping level 4.3 is primarily for use with DFHJS2LS, but is also included in the CICS web services assistants, XML assistants, and JSON assistants. This mapping level implements support for multidimensional arrays in JSON.

## Mapping level 4.2

Mapping level 4.2 is compatible with CICS TS V5.4 with APAR PI86039 applied, and higher.

Mapping level 4.2 is primarily for use with DFHJS2LS, but is also included in the CICS web services assistants, XML assistants, and JSON assistants. This mapping level implements support for Additional Properties in JSON, and introduces the following three parameters to DFHJS2LS: **ADDITIONAL-PROPERTIES-DEFAULT**, **ADDITIONAL-PROPERTIES-MAX**, and **ADDITIONAL-PROPERTIES-SIZE**.

## Mapping level 4.1

Mapping level 4.1 is compatible with a CICS TS 5.3 region and higher.

Mapping level 4.1 is added to the CICS web services assistants, XML assistants, and JSON assistants. This mapping level implements improved mappings for simple arrays generated bottom-up from existing copybooks; it also adds the ability for CICS to auto-detect uninitialized trailing storage in arrays, and to omit those records from the generated XML/JSON form. For more information, see [“COBOL to JSON schema mapping”](#) on page 257.

DFHLS2WS, DFHWS2LS, DFHLS2SC, DFHSC2LS, DFHJS2LS, and DFHLS2JS support the **TRUNCATE-NULL-ARRAYS** and **TRUNCATE-NULL-ARRAY-VALUES** parameters.

If you specify any value for **TRUNCATE-NULL-ARRAY-VALUES**, you must also specify **TRUNCATE-NULL-ARRAYS=ENABLED**.

## Mapping level 4.0

Mapping level 4.0 is compatible with a CICS TS 5.3 region.

At mapping level 4.0 and higher, DFHLS2SC and DFHLS2WS support the COBOL OCCURS DEPENDING ON clause and supports mapping of COBOL character arrays into XML strings. You can set this behavior by using the **CHAR-OCCURS** parameter on the CICS assistants.

- You must specify the parameter **DATA-TRUNCATION=ENABLED**.
- Complex OCCURS DEPENDING ON is not supported. This limitation means that OCCURS DEPENDING ON is only supported for the last field of a structure.
- CICS does not support qualified names (using the 'OF' keyword) as the target of an OCCURS DEPENDING ON clause, for example FIELD1 OF STRUCTURE1.
- CICS does not support the UNBOUNDED keyword. You must specify the maximum size of the table that is expected by the application.

**Note:** DFHLS2WS and DFHLS2SC do not support the COBOL GROUP USAGE NATIONAL clause.

## Mapping level 3.0 and higher

Mapping level 3.0 is compatible with a CICS region at any in-service level.

This mapping level provides the following support:

- DFHSC2LS and DFHWS2LS map `xsd:date` 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 region at any in-service level.

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 region at any in-service level.

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 about mapping variably repeating content inline, see [Variable arrays of elements](#).

Support for the `minOccurs` attribute is enhanced for the XML schema <sequence>, <choice>, and <all> elements. If `minOccurs="0"`, the CICS assistant treats these elements 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 you are 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 region at any in-service level.

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, the following rules apply after the value of this parameter is used to calculate the amount of space that is 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 that is associated with them, you can assign a default length by 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 region at any in-service level.

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 that are 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 that is 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 on validation 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. For more information, see [Support for XML attributes](#).
- The `xsi:nil` attribute is supported. For more information, see [Support for XML attributes](#).

## Mapping level 1.1 only

Mapping level 1.1 is compatible with a CICS region at any in-service level.

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 that are defined with different values in the XML schema. Data is handled in the following ways:

- Variable-length binary data types map to a container. A 16-byte field is created in the language structure to store the name of the container. At run time, the binary data is stored in a container and the container name is put in the language structure.
- Variable-length character data types that have a maximum length 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 by using the formula:  $4 \times (\text{ceil}(z / 3))$  where:
  - $z$  is the length of the data type in the XML schema.
  - $\text{ceil}(x)$  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 must 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 region at any in-service level.

Note the following limitations, which are 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 example:

```
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.

## Learn more

- [“How the CICS assistants map between high-level languages and XML schemas” on page 329](#)
- [“COBOL to JSON schema mapping” on page 257](#)
- [“JSON schema to COBOL mapping” on page 264](#)
- [“C and C++ to JSON schema mapping” on page 272](#)
- [“JSON schema to C and C++ mapping” on page 275](#)
- [“PL/I to JSON schema mapping” on page 282](#)
- [“JSON schema to PL/I mapping” on page 287](#)

## COBOL to XML schema mapping

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

### How COBOL names are converted to XML

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`.

### How COBOL data description elements are mapped to XML

CICS maps COBOL data description elements to schema elements according to [Table 14 on page 338](#).

#### Restriction:

- COBOL data description elements that are not shown in [Table 14 on page 338](#) are not supported by DFHLS2SC or DFHLS2WS.
- 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:
  - 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 that are supported for DISPLAY and COMPUTATIONAL-5 data description items are 9, S, and Z.
- The PICTURE characters that are supported for PACKED-DECIMAL data description items are 9, S, V, and Z.
- The only PICTURE characters that are 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.
- The OCCURS DEPENDING ON clause is supported at a mapping level of 4.0 or higher. Complex OCCURS DEPENDING ON is not supported. This means that OCCURS DEPENDING ON is only supported for the last field of a structure.
- The OCCURS INDEXED BY clause is supported at any mapping level.
- The OCCURS clause is supported up to 65535 TIMES of occurrences. This means OCCURS *n* TIMES where *n* is greater than 65535 is not supported.

Table 14. Mapping reference for COBOL data description elements	
COBOL data description	Schema simpleType
PIC X( <i>n</i> ) PIC A( <i>n</i> ) PIC G( <i>n</i> ) DISPLAY-1 PIC N( <i>n</i> )	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:maxLength value="<i>n</i>"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:short"&gt;     &lt;xsd:minInclusive value="-<i>n</i>"/&gt;     &lt;xsd:maxInclusive value="<i>n</i>"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>

Table 14. Mapping reference for COBOL data description elements (continued)

COBOL data description	Schema simpleType
PIC S9( <i>z</i> ) DISPLAY where $5 \leq z \leq 9$	<pre data-bbox="699 258 1138 394">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:int"&gt;     &lt;xsd:minInclusive value="-n"/&gt;     &lt;xsd:maxInclusive value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 426 1422 485">where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC S9( <i>z</i> ) DISPLAY where $9 < z$	<pre data-bbox="699 537 1149 674">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:long"&gt;     &lt;xsd:minInclusive value="-n"/&gt;     &lt;xsd:maxInclusive value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 705 1422 764">where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<pre data-bbox="699 816 1263 953">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedShort"&gt;     &lt;xsd:minInclusive value="0"/&gt;     &lt;xsd:maxInclusive value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 984 1422 1043">where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9( <i>z</i> ) DISPLAY where $5 \leq z \leq 9$	<pre data-bbox="699 1096 1239 1232">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedInt"&gt;     &lt;xsd:minInclusive value="0"/&gt;     &lt;xsd:maxInclusive value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 1264 1422 1323">where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC 9( <i>z</i> ) DISPLAY where $9 < z$	<pre data-bbox="699 1375 1250 1512">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedLong"&gt;     &lt;xsd:minInclusive value="0"/&gt;     &lt;xsd:maxInclusive value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 1543 1422 1602">where <i>n</i> is the maximum value that can be represented by the pattern of '9' characters.</p>
PIC S9( <i>n</i> ) COMP PIC S9( <i>n</i> ) COMP-4 PIC S9( <i>n</i> ) COMP-5 PIC S9( <i>n</i> ) BINARY where $n \leq 4$ .	<pre data-bbox="699 1654 1162 1738">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:short"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

Table 14. Mapping reference for COBOL data description elements (continued)

COBOL data description	Schema simpleType
PIC S9( <i>n</i> ) COMP PIC S9( <i>n</i> ) COMP-4 PIC S9( <i>n</i> ) COMP-5 PIC S9( <i>n</i> ) BINARY  where $5 \leq n \leq 9$ .	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:int"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
PIC S9( <i>n</i> ) COMP PIC S9( <i>n</i> ) COMP-4 PIC S9( <i>n</i> ) COMP-5 PIC S9( <i>n</i> ) BINARY  where $9 < n$ .	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:long"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
PIC 9( <i>n</i> ) COMP PIC 9( <i>n</i> ) COMP-4 PIC 9( <i>n</i> ) COMP-5 PIC 9( <i>n</i> ) BINARY  where $n \leq 4$ .	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedShort"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
PIC 9( <i>n</i> ) COMP PIC 9( <i>n</i> ) COMP-4 PIC 9( <i>n</i> ) COMP-5 PIC 9( <i>n</i> ) BINARY  where $5 \leq n \leq 9$ .	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedInt"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
PIC 9( <i>n</i> ) COMP PIC 9( <i>n</i> ) COMP-4 PIC 9( <i>n</i> ) COMP-5 PIC 9( <i>n</i> ) BINARY  where $9 < n$ .	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedLong"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
PIC S9( <i>m</i> )V9( <i>n</i> ) COMP-3          where $p = m + n$ .	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:decimal"&gt;     &lt;xsd:totalDigits value="p"/&gt;     &lt;xsd:fractionDigits value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

Table 14. Mapping reference for COBOL data description elements (continued)

COBOL data description	Schema simpleType
<p>PIC 9( <i>m</i> )V9( <i>n</i> ) COMP-3</p> <p>PIC S9( <i>m</i> ) COMP-3</p> <p>Supported at mapping level 3.0 when DATETIME=PACKED15</p>	<pre data-bbox="699 258 1187 415">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:decimal"&gt;     &lt;xsd:totalDigits value="p"/&gt;     &lt;xsd:fractionDigits value="n"/&gt;     &lt;xsd:minInclusive value="0"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 447 883 474">where <math>p = m + n</math>.</p> <pre data-bbox="699 520 1198 615">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:dateTime"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 646 1252 674">The format of the time stamp is CICS ABSTIME.</p>
<p>PIC S9( <i>m</i> )V9( <i>n</i> ) DISPLAY</p> <p>Supported at mapping level 1.2 and higher</p>	<pre data-bbox="699 726 1187 863">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:decimal"&gt;     &lt;xsd:totalDigits value="p"/&gt;     &lt;xsd:fractionDigits value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="683 894 883 921">where <math>p = m + n</math>.</p>
<p>COMP-1</p> <p>Supported at mapping level 1.2 and higher</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:float</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-1 data types with fixed precision alternatives.</p>	<pre data-bbox="699 972 1162 1066">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

Table 14. Mapping reference for COBOL data description elements (continued)

COBOL data description	Schema simpleType
<p>COMP-2</p> <p>Supported at mapping level 1.2 and higher</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:double</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-2 data types with fixed precision alternatives.</p>	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
<p><i>data description</i> OCCURS <i>n</i> TIMES</p>	<pre>&lt;xsd:element   name="field-name"   minOccurs="n"   maxOccurs="n"&gt;   ... &lt;/xsd:element&gt;</pre> <p>The content of the element depends on the data type used.</p>
<p><i>data description</i> OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDING ON <i>t</i></p> <p>Supported at mapping level 4.0</p>	<pre>&lt;xsd:element   name="field-name"   minOccurs="n"   maxOccurs="m"&gt;   ... &lt;/xsd:element&gt;</pre>
<p>PIC X OCCURS <i>n</i> TIMES PIC A OCCURS <i>n</i> TIMES PIC G DISPLAY-1 OCCURS <i>n</i> TIMES PIC N OCCURS <i>n</i> TIMES</p>	<p>When <b>CHAR-OCCURS</b> =STRING :</p> <pre>&lt;xsd:element name="field-name"&gt;   &lt;xsd:simpleType&gt;     &lt;xsd:restriction base="s:string"&gt;       &lt;xsd:maxLength value="n"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt; &lt;/xsd:element&gt;</pre> <p>This is a string.</p> <p>When <b>CHAR-OCCURS</b> =ARRAY :</p> <pre>&lt;xsd:element   name="field-name"   minOccurs="n"   maxOccurs="n"&gt;   &lt;xsd:simpleType&gt;     &lt;xsd:restriction base="s:string"&gt;       &lt;xsd:maxLength value="1"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt; &lt;/xsd:element&gt;</pre> <p>This is an array of single characters.</p>

Table 14. Mapping reference for COBOL data description elements (continued)

COBOL data description	Schema simpleType
<p>PIC X OCCURS <i>n</i> TO <i>m</i> TIMES            DEPENDING ON <i>t</i></p> <p>PIC A OCCURS <i>n</i> TO <i>m</i> TIMES            DEPENDING ON <i>t</i></p> <p>PIC G DISPLAY-1 OCCURS <i>n</i> TO <i>m</i> TIMES            DEPENDING ON <i>t</i></p> <p>PIC N OCCURS <i>n</i> TO <i>m</i> TIMES            DEPENDING ON <i>t</i></p>	<p>When <b>CHAR-OCCURS</b> =STRING :</p> <pre data-bbox="699 302 1175 485" style="background-color: #f0f0f0;"> &lt;xsd:element name="field-name"&gt;   &lt;xsd:simpleType&gt;     &lt;xsd:restriction base="s:string"&gt;       &lt;xsd:maxLength value="m"&gt;       &lt;xsd:minLength value="n"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt; &lt;/xsd:element&gt;           </pre>
<p>PIC N(<i>n</i>) USAGE NATIONAL            When <b>CHAR-USAGE</b> =NATIONAL : PIC            N(<i>n</i>)</p>	<pre data-bbox="699 810 1187 947" style="background-color: #f0f0f0;"> &lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:maxLength value="n"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;           </pre> <p>At run time, CICS populates the application data structure field with UTF-16 data.</p>

## XML schema to COBOL mapping

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

### How XML schema element names are mapped to COBOL

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.

- 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`.

Should the above behavior be undesirable the user can specify `MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING` as input to the utility which disables the addition of one or two numeric digits to the second and subsequent instances of the name.

- 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 368.

- 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 374.

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.

## How XML schema types are mapped to COBOL

DFHSC2LS and DFHWS2LS map schema types to COBOL data description elements by using the specified mapping level according to [Table 15 on page 344](#). 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
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:anyType"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	Mapping level 2.0 and below: Not supported Mapping level 2.1: Supported
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:anySimpleType"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	Mapping level 1.0: Not supported Mapping level 1.1 and higher:  PIC X(255)

Table 15. Mapping reference for XML schema types (continued)

Schema simple type	COBOL data description
<pre data-bbox="167 258 613 369">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 401 397 428">where <i>type</i> is one of:</p> <ul data-bbox="188 449 444 831" style="list-style-type: none"> <li>string</li> <li>normalizedString</li> <li>token</li> <li>Name</li> <li>NMTOKEN</li> <li>language</li> <li>NCName</li> <li>ID</li> <li>IDREF</li> <li>ENTITY</li> <li>hexBinary</li> </ul>	<p data-bbox="823 245 1045 273">All mapping levels:</p> <pre data-bbox="834 302 935 327">PIC X(z)</pre>
<pre data-bbox="167 882 613 993">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 1024 397 1052">where <i>type</i> is one of:</p> <ul data-bbox="188 1073 444 1419" style="list-style-type: none"> <li>string</li> <li>normalizedString</li> <li>token</li> <li>Name</li> <li>NMTOKEN</li> <li>language</li> <li>NCName</li> <li>ID</li> <li>IDREF</li> <li>ENTITY</li> </ul>	<p data-bbox="823 869 1442 896">When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <pre data-bbox="834 926 1127 951">PIC N(z) USAGE NATIONAL</pre>
<pre data-bbox="167 1472 613 1562">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 1593 397 1621">where <i>type</i> is one of:</p> <ul data-bbox="188 1642 347 1913" style="list-style-type: none"> <li>duration</li> <li>date</li> <li>time</li> <li>gDay</li> <li>gMonth</li> <li>gYear</li> <li>gMonthDay</li> <li>gYearMonth</li> </ul>	<p data-bbox="823 1459 1045 1486">All mapping levels:</p> <pre data-bbox="834 1516 948 1541">PIC X(32)</pre>

Table 15. Mapping reference for XML schema types (continued)

Schema simple type	COBOL data description
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:dateTime"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>Mapping level 1.2 and below:</p> <p>PIC X(32)</p> <p>Mapping level 2.0 and higher:</p> <p>PIC X(40)</p> <p>Mapping level 3.0 and higher:</p> <p>PIC S9(15) COMP-3</p> <p>The format is CICS ABSTIME.</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>where <i>type</i> is one of:</p> <p>byte</p> <p>unsignedByte</p>	<p>All mapping levels:</p> <p>PIC X DISPLAY</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:short"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC S9999 COMP-5 SYNC</p> <p>or</p> <p>PIC S9999 DISPLAY</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedShort"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC 9999 COMP-5 SYNC</p> <p>or</p> <p>PIC 9999 DISPLAY</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:integer"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC S9(18) COMP-3</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:int"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC S9(9) COMP-5 SYNC</p> <p>or</p> <p>PIC S9(9) DISPLAY</p>

Table 15. Mapping reference for XML schema types (continued)

Schema simple type	COBOL data description
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedInt"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC 9(9) COMP-5 SYNC</p> <p>or</p> <p>PIC 9(9) DISPLAY</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:long"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC S9(18) COMP-5 SYNC</p> <p>or</p> <p>PIC S9(18) DISPLAY</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedLong"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC 9(18) COMP-5 SYNC</p> <p>or</p> <p>PIC 9(18) DISPLAY</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:decimal"&gt;     &lt;xsd:totalDigits value="m"&gt;     &lt;xsd:fractionDigits value="n"&gt;     &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>If WIDE-COMP3=FULL:</p> <p>PIC 9(m)V9(n) COMP-3</p> <p>Otherwise:</p> <p>PIC 9(p)V9(n) COMP-3</p> <p>where <math>p = m - n</math>.</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:boolean"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>PIC X DISPLAY</p> <p>The value x'00' implies false, x'01' implies true.</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:list&gt;     &lt;xsd:simpleType&gt;       &lt;xsd:restriction base="xsd:int"/&gt;     &lt;/xsd:simpleType&gt;   &lt;/xsd:list&gt; &lt;/xsd:simpleType&gt;</pre>	<p>Mapping level 1.0: Not supported</p> <p>Mapping level 1.1 and higher:</p> <p>PIC X(255)</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:union memberTypes="xsd:int xsd:string"/&gt; &lt;/xsd:simpleType&gt;</pre>	<p>Mapping level 1.0: Not supported</p> <p>Mapping level 1.1 and higher:</p> <p>PIC X(255)</p>

Table 15. Mapping reference for XML schema types (continued)

Schema simple type	COBOL data description
<pre data-bbox="167 258 716 369">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:base64Binary"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <pre data-bbox="167 415 716 506">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:base64Binary"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 533 526 562">where the length is not defined.</p>	<p data-bbox="821 245 1214 275">Mapping level 1.0: Not supported</p> <p data-bbox="821 296 1036 325">Mapping level 1.1:</p> <pre data-bbox="834 352 935 382">PIC X(y)</pre> <p data-bbox="821 409 1463 470">where <math>y = 4 \times (\text{ceil}(z/3))</math>. <math>\text{ceil}(x)</math> is the smallest integer greater than or equal to <math>x</math>.</p> <p data-bbox="821 491 1170 520">Mapping level 1.2 and higher:</p> <pre data-bbox="834 548 935 577">PIC X(z)</pre> <p data-bbox="821 604 1117 634">where the length is fixed.</p> <pre data-bbox="834 661 948 690">PIC X(16)</pre> <p data-bbox="821 718 1446 814">where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.</p>
<pre data-bbox="167 863 626 953">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="821 848 1166 877">Mapping level 1.1 and below:</p> <pre data-bbox="834 905 948 934">PIC X(32)</pre> <p data-bbox="821 961 1170 991">Mapping level 1.2 and higher:</p> <pre data-bbox="834 1018 915 1047">COMP-1</pre> <p data-bbox="821 1075 1446 1390"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:float</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-1 data types with fixed precision alternatives.</p>

Table 15. Mapping reference for XML schema types (continued)

Schema simple type	COBOL data description
<pre data-bbox="167 258 639 348">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="823 245 1166 275">Mapping level 1.1 and below:</p> <pre data-bbox="834 302 948 327">PIC X(32)</pre> <p data-bbox="823 359 1166 388">Mapping level 1.2 and higher:</p> <pre data-bbox="834 415 915 441">COMP-2</pre> <p data-bbox="823 472 1442 789"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:double</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of COMP-2 data types with fixed precision alternatives.</p>

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.

### How C and C++ names are converted to XML

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`.

### How C and C++ data types are mapped to XML

DFHLS2SC and DFHLS2WS map C and C++ data types to schema elements according to [Table 16 on page 351](#).

The `_Packed` qualifier is supported for structures.

**Restriction:**

- C and C++ types that are not shown in [Table 16 on page 351](#) are not supported by DFHLS2SC or DFHLS2WS.
- 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 ]	<pre data-bbox="613 485 1084 600">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
char16_t[ n ]	<p data-bbox="597 646 980 674">At mapping level 4.0 and higher:</p> <pre data-bbox="613 709 1097 846">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:maxLength value="n"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="597 877 1403 936">At run time, CICS populates the application data structure field with UTF-16 data.</p>
char[8] Supported at mapping level 3.0 and higher when DATETIME=PACKED15	<pre data-bbox="613 989 1110 1079">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:dateTime"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="597 1104 1159 1131">The format of the time stamp is CICS ABSTIME.</p>
char	<pre data-bbox="613 1178 1058 1268">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:byte"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
unsigned char	<pre data-bbox="613 1325 1162 1415">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedByte"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
short	<pre data-bbox="613 1472 1071 1562">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:short"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
unsigned short	<pre data-bbox="613 1619 1175 1709">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedShort"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
int long	<pre data-bbox="613 1766 1045 1856">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:int"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

Table 16. Mapping reference for C and C++ data types (continued)

C and C++ data type	Schema simpleType
unsigned int unsigned long	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedInt"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
long long	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:long"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
unsigned long long	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedLong"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
bool (C++ only)	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:boolean"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
float Supported at mapping level 1.2 and higher	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:float</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of float data types with fixed precision alternatives.</p>
double Supported at mapping level 1.2 and higher	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:double</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of double data types with fixed precision alternatives.</p>

## 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.

### How XML schema element names are converted to C and C++

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 368.

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.

### How XML schema types are mapped to C and C++

DFHSC2LS and DFHWS2LS map schema types to C and C++ data types according to [Table 17 on page 353](#). 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
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:anyType"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	Mapping level 2.0 and below: Not supported Mapping level 2.1 and higher: Supported
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:anySimpleType"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	Mapping level 1.0: Not supported Mapping level 1.1 and higher:  <pre>char[255]</pre>

Table 17. Mapping reference for XML schema types (continued)

Schema simpleType	C and C++ data type
<pre data-bbox="167 258 613 373">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 401 397 428">where <i>type</i> is one of:</p> <ul data-bbox="188 449 444 835" style="list-style-type: none"> <li>string</li> <li>normalizedString</li> <li>token</li> <li>Name</li> <li>NMTOKEN</li> <li>language</li> <li>NCName</li> <li>ID</li> <li>IDREF</li> <li>ENTITY</li> <li>hexBinary</li> </ul>	<p data-bbox="823 245 1045 273">All mapping levels:</p> <pre data-bbox="836 302 922 329">char[z]</pre>
<pre data-bbox="167 882 613 997">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 1024 397 1052">where <i>type</i> is one of:</p> <ul data-bbox="188 1073 444 1421" style="list-style-type: none"> <li>string</li> <li>normalizedString</li> <li>token</li> <li>Name</li> <li>NMTOKEN</li> <li>language</li> <li>NCName</li> <li>ID</li> <li>IDREF</li> <li>ENTITY</li> </ul>	<p data-bbox="823 869 1443 896">When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <pre data-bbox="836 926 971 953">char16_t[z]</pre>

Table 17. Mapping reference for XML schema types (continued)

Schema simpleType	C and C++ data type
<pre data-bbox="167 258 613 348">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 380 399 407">where <i>type</i> is one of:</p> <ul data-bbox="188 426 350 737" style="list-style-type: none"> <li>duration</li> <li>date</li> <li>decimal</li> <li>time</li> <li>gDay</li> <li>gMonth</li> <li>gYear</li> <li>gMonthDay</li> <li>gYearMonth</li> </ul>	<p data-bbox="816 247 1045 275">All mapping levels:</p> <pre data-bbox="833 306 938 333">char[32]</pre>
<pre data-bbox="167 789 667 879">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:dateTime"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 779 1170 806">Mapping level 1.2 and below:</p> <pre data-bbox="833 837 938 865">char[32]</pre> <p data-bbox="816 896 1170 924">Mapping level 2.0 and higher:</p> <pre data-bbox="833 955 938 982">char[40]</pre> <p data-bbox="816 1014 1170 1041">Mapping level 3.0 and higher:</p> <pre data-bbox="833 1073 938 1100">char[8]</pre> <p data-bbox="816 1131 1386 1159">The format of the time stamp is CICS ABSTIME.</p>
<pre data-bbox="167 1220 613 1310">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:byte"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 1188 1045 1215">All mapping levels:</p> <pre data-bbox="833 1247 976 1274">signed char</pre>
<pre data-bbox="167 1373 716 1463">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedByte"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 1362 1045 1390">All mapping levels:</p> <pre data-bbox="833 1421 889 1449">char</pre>
<pre data-bbox="167 1524 626 1614">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:short"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 1514 1045 1541">All mapping levels:</p> <pre data-bbox="833 1572 906 1600">short</pre>
<pre data-bbox="167 1675 732 1766">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedShort"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 1665 1045 1692">All mapping levels:</p> <pre data-bbox="833 1724 1016 1751">unsigned short</pre>
<pre data-bbox="167 1827 651 1917">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:integer"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 1816 1045 1843">All mapping levels:</p> <pre data-bbox="833 1875 938 1902">char[33]</pre>

Table 17. Mapping reference for XML schema types (continued)

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

Table 17. Mapping reference for XML schema types (continued)

Schema simpleType	C and C++ data type
<pre data-bbox="167 258 716 369">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:base64Binary"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <pre data-bbox="167 415 716 506">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:base64Binary"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 533 521 562">where the length is not defined</p>	<p data-bbox="821 245 1166 275">Mapping level 1.1 and below:</p> <pre data-bbox="834 302 922 331">char[y]</pre> <p data-bbox="821 359 1463 420">where <math>y = 4 \times (\text{ceil}(z / 3))</math>. <math>\text{ceil}(x)</math> is the smallest integer greater than or equal to <math>x</math>.</p> <p data-bbox="821 436 1170 466">Mapping level 1.2 and higher:</p> <pre data-bbox="834 493 922 522">char[z]</pre> <p data-bbox="821 550 1117 579">where the length is fixed.</p> <pre data-bbox="834 606 935 636">char[16]</pre> <p data-bbox="821 663 1414 724">is the name of the container that stores the binary data when the length is not defined.</p>
<pre data-bbox="167 774 626 865">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="821 762 1166 791">Mapping level 1.1 and below:</p> <pre data-bbox="834 819 935 848">char[32]</pre> <p data-bbox="821 875 1170 905">Mapping level 1.2 and higher:</p> <pre data-bbox="834 932 935 961">float(*)</pre> <p data-bbox="821 989 1446 1306"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:float</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of float data types with fixed precision alternatives.</p>
<pre data-bbox="167 1358 639 1449">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="821 1346 1166 1375">Mapping level 1.0 and below:</p> <pre data-bbox="834 1402 935 1432">char[32]</pre> <p data-bbox="821 1459 1170 1488">Mapping level 1.2 and higher:</p> <pre data-bbox="834 1516 948 1545">double(*)</pre> <p data-bbox="821 1572 1446 1890"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:double</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of double data types with fixed precision alternatives.</p>

## 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.

### How PL/I names are converted to XML

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`.

### How PL/I data types are mapped to XML

DFHLS2SC and DFHLS2WS map PL/I data types to schema elements according to [Table 18 on page 359](#).

#### Restriction:

- PL/I types that are not shown in [Table 18 on page 359](#) are not supported by DFHLS2SC or DFHLS2WS.
- 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 that are specified as DECIMAL( $p$ ,  $q$ ) are supported only when  $p \geq q$
- Data items that are 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 doubleword boundary and each byte in the structure must be mapped to the correct boundary.

For example, 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 fullword boundary.
- FIELD3 is 8 bytes long and must be aligned on a doubleword 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 more information about PL/I structural memory alignment requirements, see [Enterprise PL/I for z/OS product information](#).

PL/I data description	Schema
FIXED BINARY ( <i>n</i> ) where $n \leq 7$	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:byte"/&gt; &lt;/xsd:simpleType&gt;</pre>
FIXED BINARY ( <i>n</i> ) where $8 \leq n \leq 15$	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:short"/&gt; &lt;/xsd:simpleType&gt;</pre>
FIXED BINARY ( <i>n</i> ) where $16 \leq n \leq 31$	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:int"/&gt; &lt;/xsd:simpleType&gt;</pre>
FIXED BINARY ( <i>n</i> ) where $32 \leq n \leq 63$ <b>Restriction:</b> Enterprise PL/I only	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:long"/&gt; &lt;/xsd:simpleType&gt;</pre>
UNSIGNED FIXED BINARY( <i>n</i> ) where $n \leq 8$ <b>Restriction:</b> Enterprise PL/I only	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedByte"/&gt; &lt;/xsd:simpleType&gt;</pre>
UNSIGNED FIXED BINARY( <i>n</i> ) where $9 \leq n \leq 16$ <b>Restriction:</b> Enterprise PL/I only	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedShort"/&gt; &lt;/xsd:simpleType&gt;</pre>
UNSIGNED FIXED BINARY( <i>n</i> ) where $17 \leq n \leq 32$ <b>Restriction:</b> Enterprise PL/I only	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedInt"/&gt; &lt;/xsd:simpleType&gt;</pre>
UNSIGNED FIXED BINARY( <i>n</i> ) where $33 \leq n \leq 64$ <b>Restriction:</b> Enterprise PL/I only	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedLong"/&gt; &lt;/xsd:simpleType&gt;</pre>

Table 18. Mapping reference for PL/I data types (continued)

PL/I data description	Schema
FIXED DECIMAL( <i>n</i> , <i>m</i> )	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:decimal"&gt;     &lt;xsd:totalDigits value="n"/&gt;     &lt;xsd:fractionDigits value="m"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
FIXED DECIMAL(15) Supported at mapping level 3.0 and higher when DATETIME=PACKED15	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:dateTime"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>The format of the time stamp is CICS ABSTIME.</p>
BIT( <i>n</i> ) where <i>n</i> is a multiple of 8. Other values are not supported.	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:hexBinary"&gt;     &lt;xsd:length value="m"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>where <math>m = n / 8</math></p>
CHARACTER( <i>n</i> ) VARYING and VARYINGZ are also supported at mapping level 1.2 and higher. <b>Restriction:</b> VARYINGZ is supported only by Enterprise PL/I	<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:maxLength value="n"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
GRAPHIC( <i>n</i> ) VARYING and VARYINGZ are also supported at mapping level 1.2 and higher. <b>Restriction:</b> VARYINGZ is supported only by Enterprise PL/I	<p>At a mapping level of 1.0 and 1.1, where <math>m = 2 * n</math> :</p> <pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:hexBinary"&gt;     &lt;xsd:length value="m"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>At a mapping level of 1.2 or higher:</p> <pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:length value="n"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

Table 18. Mapping reference for PL/I data types (continued)

PL/I data description	Schema
<p>WIDECHAR( <i>n</i> )</p> <p><b>Restriction:</b> Enterprise PL/I only</p>	<p>At a mapping level of 1.0 and 1.1, where <math>m = 2 * n</math> :</p> <pre data-bbox="831 289 1351 415">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:hexBinary"&gt;     &lt;xsd:length value="m"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>At a mapping level of 1.2 or higher:</p> <pre data-bbox="831 499 1351 625">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:hexBinary"&gt;     &lt;xsd:length value="n"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>At mapping level 4.0 and higher, CICS populates the application data structure field with UTF-16 data.</p> <pre data-bbox="831 739 1318 886">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:maxLength value="n"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
<p>ORDINAL</p> <p><b>Restriction:</b> Enterprise PL/I only</p>	<pre data-bbox="831 936 1295 1012">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:byte"/&gt; &lt;/xsd:simpleType&gt;</pre>
<p>BINARY FLOAT( <i>n</i> )</p> <p>where <math>n \leq 21</math></p> <p>Supported at mapping level 1.2 and higher.</p>	<pre data-bbox="831 1062 1295 1159">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
<p>BINARY FLOAT( <i>n</i> )</p> <p>where <math>21 &lt; n \leq 53</math></p> <p>Values greater than 53 are not supported.</p> <p>Supported at mapping level 1.2 and higher.</p>	<pre data-bbox="831 1222 1312 1318">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
<p>DECIMAL FLOAT( <i>n</i> )</p> <p>where <math>n \leq 6</math></p> <p>Supported at mapping level 1.2 and higher.</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:float</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>	<pre data-bbox="831 1419 1295 1516">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

Table 18. Mapping reference for PL/I data types (continued)

PL/I data description	Schema
<p>DECIMAL FLOAT( <i>n</i> ) where <math>6 &lt; n \leq 16</math>                      Values greater than 16 are not supported.                      Supported at mapping level 1.2 and higher.</p> <p><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:double</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>	<pre data-bbox="833 258 1312 352">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>

## 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.

## How XML schema element names are converted to PL/I

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`.

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

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`.

Should the above behavior be undesirable the user can specify **MAPPING-OVERRIDES=NO-ARRAY-NAME-INDEXING** as input to the utility which disables the addition of one or two numeric digits to the second and subsequent instances of the name.

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 368](#).

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 374](#).

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.

## How XML schema types are mapped to PL/I

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
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:anyType"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	Mapping level 2.0 and below: Not supported Mapping level 2.1 and higher: Supported
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:anySimpletype"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	Mapping level 1.1 and higher: CHAR(255)
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;     &lt;xsd:maxLength value="z"/&gt;     &lt;xsd:whiteSpace value="preserve"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>where <i>type</i> is one of:</p> <ul style="list-style-type: none"> <li>string</li> <li>normalizedString</li> <li>token</li> <li>Name</li> <li>NMTOKEN</li> <li>language</li> <li>NCName</li> <li>ID</li> <li>IDREF</li> <li>ENTITY</li> </ul>	All mapping levels: CHARACTER( z )

Table 19. Mapping reference for XML schema types (continued)

Schema	PL/I data description
<pre data-bbox="167 258 613 373">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;     &lt;xsd:length value="z"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 401 397 428">where <i>type</i> is one of:</p> <ul data-bbox="188 449 444 793" style="list-style-type: none"> <li>string</li> <li>normalizedString</li> <li>token</li> <li>Name</li> <li>NMTOKEN</li> <li>language</li> <li>NCName</li> <li>ID</li> <li>IDREF</li> <li>ENTITY</li> </ul>	<p data-bbox="823 245 1442 273">When <b>CCSID=1200</b> at mapping level 4.0 and higher:</p> <p data-bbox="834 302 976 329">WIDECHAR(<i>z</i>)</p>
<pre data-bbox="167 846 613 940">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:type"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p data-bbox="151 968 397 995">where <i>type</i> is one of:</p> <ul data-bbox="188 1016 347 1297" style="list-style-type: none"> <li>duration</li> <li>date</li> <li>time</li> <li>gDay</li> <li>gMonth</li> <li>gYear</li> <li>gMonthDay</li> <li>gYearMonth</li> </ul>	<p data-bbox="823 833 1045 861">All mapping levels:</p> <p data-bbox="823 884 943 911">CHAR(32)</p>
<pre data-bbox="167 1344 667 1438">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:dateTime"&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="823 1331 1167 1358">Mapping level 1.2 and below:</p> <p data-bbox="834 1388 937 1415">CHAR(32)</p> <p data-bbox="823 1444 1170 1472">Mapping level 2.0 and higher:</p> <p data-bbox="834 1501 937 1528">CHAR(40)</p> <p data-bbox="823 1558 1170 1585">Mapping level 3.0 and higher:</p> <p data-bbox="834 1614 1050 1642">FIXED DECIMAL(15)</p> <p data-bbox="823 1671 1382 1698">The format of the time stamp is CICS ABSTIME.</p>

Table 19. Mapping reference for XML schema types (continued)

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

Table 19. Mapping reference for XML schema types (continued)

Schema	PL/I data description
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedInt"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p><b>Enterprise PL/I</b> UNSIGNED FIXED BINARY(32)</p> <p><b>Other PL/I</b> BIT(64)</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:long"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p>Mapping level 1.1 and below:</p> <p><b>Enterprise PL/I</b> SIGNED FIXED BINARY(63)</p> <p><b>Note:</b> The LIMITS compiler directive might affect how the PL/I compiler interprets this field. CICS expects the field to be of the declared size, but the compiler might optimize the field into a smaller space, which can result in a mismatch. To avoid such problems, use the LIMITS(FIXEDBIN(63)) compile-time option.</p> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b> CHAR( y )</p> <p>where y is a fixed length that is less than 16 MB.</p> <p>All mapping levels:</p> <p><b>Other PL/I</b> BIT(64)</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:unsignedLong"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p>Mapping level 1.1 and below:</p> <p><b>Enterprise PL/I</b> UNSIGNED FIXED BINARY(64)</p> <p><b>Note:</b> The LIMITS compiler directive might affect how the PL/I compiler interprets this field. CICS expects the field to be of the declared size, but the compiler might optimize the field into a smaller space, which can result in a mismatch. To avoid such problems, use the LIMITS(FIXEDBIN(63)) compile-time option.</p> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b> CHAR( y )</p> <p>where y is a fixed length that is less than 16 MB.</p> <p>All mapping levels:</p> <p><b>Other PL/I</b> BIT(64)</p>

Table 19. Mapping reference for XML schema types (continued)

Schema	PL/I data description
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:boolean"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p>Mapping level 1.1 and below:</p> <p><b>Enterprise PL/I</b> SIGNED FIXED BINARY (7)</p> <p><b>Other PL/I</b> FIXED BINARY (7)</p> <p>Mapping level 1.2 and higher:</p> <p><b>Enterprise PL/I</b> BIT(7) BIT(1)</p> <p><b>Other PL/I</b> BIT(7) BIT(1)</p> <p>where BIT(7) is provided for alignment and BIT(1) contains the Boolean mapped value.</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:decimal"&gt;     &lt;xsd:totalDigits value="n"/&gt;     &lt;xsd:fractionDigits value="m"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>FIXED DECIMAL( <i>n</i> , <i>m</i> )</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:list&gt;     &lt;xsd:simpleType&gt;       &lt;xsd:restriction base="xsd:int"/&gt;     &lt;/xsd:simpleType&gt;   &lt;/xsd:list&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>CHAR(255)</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:union memberTypes="xsd:int xsd:string"/&gt; &lt;/xsd:simpleType&gt;</pre>	<p>All mapping levels:</p> <p>CHAR(255)</p>
<pre>&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:base64Binary"&gt;     &lt;xsd:length value="y"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre> <p>where the length is not defined</p>	<p>Mapping level 1.0: Not supported</p> <p>Mapping level 1.1:</p> <p>CHAR(<i>z</i>)</p> <p>where <math>z = 4 \times (\text{ceil}(y/3))</math>. <math>\text{ceil}(x)</math> is the smallest integer greater than or equal to <math>x</math>.</p> <p>Mapping level 1.2 and higher:</p> <p>CHAR(<i>y</i>)</p> <p>where the length is fixed.</p> <p>CHAR(16)</p> <p>where the length is not defined. The field holds the 16-byte name of the container that stores the binary data.</p>

Table 19. Mapping reference for XML schema types (continued)

Schema	PL/I data description
<pre data-bbox="167 258 626 348">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:float"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 247 1146 275">Mapping levels 1.0 and 1.1:</p> <p data-bbox="833 302 935 329">CHAR(32)</p> <p data-bbox="816 363 1170 390">Mapping level 1.2 and higher:</p> <p data-bbox="816 407 1247 468"><b>Enterprise PL/I</b> DECIMAL FLOAT(6) HEXADEC</p> <p data-bbox="816 485 1117 546"><b>Other PL/I</b> DECIMAL FLOAT(6)</p> <p data-bbox="816 562 1442 909"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:float</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>
<pre data-bbox="167 957 639 1050">&lt;xsd:simpleType&gt;   &lt;xsd:restriction base="xsd:double"&gt;     &lt;/xsd:restriction&gt;   &lt;/xsd:simpleType&gt;</pre>	<p data-bbox="816 947 1146 974">Mapping levels 1.0 and 1.1:</p> <p data-bbox="833 1001 935 1029">CHAR(32)</p> <p data-bbox="816 1062 1170 1089">Mapping level 1.2 and higher:</p> <p data-bbox="816 1106 1263 1167"><b>Enterprise PL/I</b> DECIMAL FLOAT(16) HEXADEC</p> <p data-bbox="816 1184 1133 1245"><b>Other PL/I</b> DECIMAL FLOAT(16)</p> <p data-bbox="816 1262 1442 1608"><b>Note:</b> The IBM Hexadecimal Floating Point (HFP) data representation is not exactly the same as the IEEE-754-1985 representation that is used for XML. Some values might not convert exactly from one representation to the other. Some extremely large or small values might not be valid for <code>xsd:double</code> data types. Some values might lose precision when converted to or from HFP representation. If precise conversions are important, consider replacing use of DECIMAL FLOAT data types with fixed precision alternatives.</p>

### 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.

#### Example: Denoting an 8-byte string that is optional

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>
```

#### Example: Denoting an 8-byte string that must occur

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

#### Example: An element that occurs exactly three times

The following 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

### Example: a mandatory element that can occur from one to five times

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 370 , 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 370](#), 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.

### Example: Nested variable arrays

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 variable-length values and white space

You can customize the way in which variable-length values and white space are handled by using settings on the CICS assistants and by adding facets directly into the XML schema.

Typically, the CICS XML assistant and the CICS Web services assistant map data strings to fixed-length character arrays; these arrays require padding with spaces or nulls. Mapping variable-length values to fixed-length data arrays can be inefficient and waste storage. If the length of your data is variable, you are recommended to customize the way these mappings are handled.

If you are converting from a language structure to an XML schema or WSDL document, you are recommended to specify the `whiteSpace` and `maxLength` facets in your XML schema and to set the **CHAR-VARYING-LIMIT** parameter on the assistants.

If you are converting from an XML schema or WSDL document to a language structure, you are recommended to set an appropriate value for the **CHAR-VARYING** parameter on the assistants.

**Note:** Null characters ('x00') are not valid in XML documents. Any null characters from application data parsed by CICS are seen to signify the end of a string and the value is truncated. When CICS generates application data it does so according to the value of the **CHAR-VARYING** parameter. For example, if the **CHAR-VARYING=NULL** option is specified, variable-length strings generated by CICS are ended with a null character.

## Mapping variable-length values from XML to language structures

Use facets in the XML schema or specify certain parameters on the CICS assistants to customize the way in which mappings between your XML schema or WSDL document and the language structure are handled.

XML data types can be restricted using facets. Use the length facets (`length`, `maxLength`, and `minLength`) and the `whiteSpace` facet to customize how variable-length data in your XML is handled.

### **length**

Used to specify that the data is of fixed length.

### **maxLength**

Used to specify the maximum length for the data type. If this value is not set for a string-based data type, the maximum length is unbounded.

### **minLength**

Used to specify the minimum length for the data type. If this value is not set for a string-based data type, the minimum length is 0.

### **whiteSpace**

Used to specify how white space around a data value is handled. White space includes spaces, tabs, and new lines. The `whiteSpace` facet can be set to `preserve`, `replace`, or `collapse`:

- A value of `preserve` maintains any white space in the data value.
- A value of `replace` means that any tabs or new lines are replaced with the appropriate number of spaces.
- A value of `collapse` means that leading, trailing, and embedded white space is removed and that all tabs, new lines, and consecutive spaces are replaced with single space characters.

If the `whiteSpace` facet is not set, white space is preserved.

For more information about XML schema facets, see [XML Schema Part 2: Datatypes Second Edition](#).

The following parameters on the CICS assistants, DFHSC2LS and DFHWS2LS, can be used to alter the way that variable-length data is mapped from the XML schema to the language structure. These parameters are available at mapping level 1.2 or higher.

### **DEFAULT-CHAR-MAXLENGTH**

Specifies the default array length of character data in characters for mappings where no length is implied in the XML schema or WSDL document. The value of this parameter can be a positive integer in the range of 1 - 2 147 483 647.

However, you are recommended to specify the maximum character length that you want DFHSC2LS or DFHWS2LS to use directly in your XML schema or WSDL document with the `maxLength` facet. Specifying the maximum length directly in the XML schema or WSDL document avoids problems associated with having one global default applied to all string-based data types.

### **CHAR-VARYING-LIMIT**

Specifies the maximum size of variable-length character data that is mapped to the language structure. If the character data is larger than the value specified in this parameter, it is mapped to

a container and the container name is used in the generated language structure. The value can range from 0 to the default 32 767 bytes.

### **CHAR-VARYING**

Specifies how variable-length character data is mapped. If you do not specify this parameter, the default mapping depends on the language specified. You can select these options:

- **CHAR-VARYING=NO** specifies that variable-length character data is mapped as fixed-length strings.
- **CHAR-VARYING=NULL** specifies that variable-length character data is mapped to null-terminated strings.
- **CHAR-VARYING=YES** specifies that 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.

Setting **CHAR-VARYING=YES** typically results in the best performance.

### **Mapping variable-length values from language structures to XML**

You can customize the way in which mappings between your language structure and the XML schema, or WSDL document are handled. Set the **CHAR-VARYING** parameter on DFHLS2SC or DFHLS2WS, to COLLAPSE or NULL to change the way that character arrays are generated.

Setting the **CHAR-VARYING=NULL** option tells CICS to add a null character at the end of each character array when generating XML.

Setting the **CHAR-VARYING=COLLAPSE** option tells CICS to automatically remove any trailing spaces from the end of character arrays when generating XML. This option is available only at mapping level 2.1 or higher and **CHAR-VARYING=COLLAPSE** is the default value at mapping level 2.1 or higher for all languages other than C and C++. When the XML is parsed, all leading, trailing, and embedded white space is removed.

### **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.

### **Example**

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.

## Example: SOAP message

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.

### Prerequisite

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.

#### **Example**

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.

## Support for <xsd:sequence>

The use of <xsd:sequence> elements with the CICS assistants is supported with a number of limitations.

The use of minOccurs and maxOccurs attributes are not supported for the <xsd:sequence> element. The exceptions to this rule are when minOccurs="0" and maxOccurs="1" or minOccurs="1" and maxOccurs="1".

The CICS assistants do not allow two elements with the same name to be used in the same <xsd:sequence> element. For example, the sequence {a, b, c, a} is rejected by the CICS Assistants. To avoid this limitation, replace the sequence with {a maxOccurs="2", b, c}.

The CICS Assistants allow only one <xsd:any> element, or one element that is treated as an <xsd:any> in the same <xsd:sequence> element. Elements that are treated as <xsd:any> elements include abstract elements that do not have a substitution group defined.

When parsing XML, CICS treats <xsd:sequence> elements like <xsd:all> elements; this means that CICS does not check whether the order of the items in the sequence is correct. For example, if the schema defines a sequence of {a, b, c}, CICS tolerates XML that contains a, b, and c in any order. However; when CICS generates XML, the order of items in an <xsd:sequence> is always preserved.

CICS does not automatically detect missing XML data. For example, if an element in the <xsd:sequence> is defined as mandatory, (minOccurs="1" maxOccurs="1"), but does not occur in the XML document, CICS will report this problem only if runtime validation is enabled

## 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.

### Example

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 [XML Schema Part 1: Structures Second Edition](#).

## 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.

### Example

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 [XML Schema Part 0: Primer Second Edition](#).

## 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.

### Example

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 copybooks. 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 [XML Schema Part 0: Primer Second Edition](#).

## Support for UTF-16 in application data

CICS web services support conversion of UTF-16 encoded application data into XML or JSON and also XML or JSON into UTF-16 encoded application data. Use UTF-16 when you need to store and process data in multiple languages.

CICS SOAP and JSON web services support conversion of UTF-16 encoded application data into XML or JSON and also XML or JSON into UTF-16 encoded application data. Unicode is a variable-width encoding scheme that enables systems to handle data efficiently.

UTF-16 is a variable width encoding for Unicode, where each character is represented by 2 or 4 bytes. CICS web services support CCSID 1200 for application data, which is UTF-16 BE (big endian) with IBM Private Use Area. This behavior is consistent with UTF-16 support in all supported languages.

UTF-16 is supported at mapping level 4.0 and upwards. You can customize how application data is converted by using mapping settings in the assistants. For more information about XML mapping levels, see [Mapping levels for the CICS assistants](#). For more information about JSON mapping levels, see [Mapping levels for the CICS JSON assistants](#).

**Note:** UTF-16 requires more processing time and is less storage efficient than EBCDIC encodings. Furthermore, mixing encoding types incurs extra runtime processing.

## Mapping UTF-16 from XML or JSON schema to language structures

Support for UTF-16 depends on how you create the web service. Mapping XML or JSON schema to language structures, also known as top-down mapping, has the following characteristics. If UTF-16 is enabled, all text fields are mapped to UTF-16 fields, whereas numeric display data types in COBOL are mapped as EBCDIC. To use UTF-16, set the CCSID parameter of DFHJS2LS, DFHSC2LS, or DFHWS2LS to 1200.

For example, if the following XML schema fragment were present in the WSDL:

```
<xsd:element name="myString" nillable="false">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="20"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The DFHWS2LS assistant generates the following field in a COBOL language structure:

```
myString PIC N(20) USAGE NATIONAL
```

The CHAR-MULTIPLIER parameter of the web services assistants can be used to specify the length of a field the assistants generate.

### CHAR-MULTIPLIER

When you use UTF-16, the only valid values for the **CHAR-MULTIPLIER** parameter are 2 or 4, where 2 is the default value.

**CHAR-MULTIPLIER** = 2 , where the schema describes a string of `maxLength x` , generates PIC `N( x )` . Setting **CHAR-MULTIPLIER** = 2 does not preclude the use of surrogate pairs in a UTF-16 string, but impacts the number of characters that fit in the field.

**CHAR-MULTIPLIER** = 4 generates PIC `N( 2x )` . If **CHAR-MULTIPLIER** = 4 , the value at run time is padded if the string includes characters that can be expressed in a single encoding unit.

## Mapping UTF-16 from language structures to XML or JSON schema

Mapping from a language structure to XML or JSON schema, also known as bottom-up mapping, is managed differently to top-down mapping. If a UTF-16 string is declared in the language structure, then the data is interpreted by CICS as UTF-16 encoded, otherwise, data is assumed to be in an EBCDIC encoding. The `CCSID` parameter for `DFHLS2JS`, `DFHLS2SC`, or `DFHLS2WS` indicates the encoding of any EBCDIC text within the application data; it must not be set to indicate UTF-16.

The data types that are interpreted as UTF-16 characters are as follows: `PIC N( n )` in COBOL, `WIDECHAR( n )` in PL/I, and `char16_t[ n ]` in C and C++.

The `CHAR-USAGE` parameter of the web services assistants can be used to specify data types.

### CHAR-USAGE

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.

If you want to mix national data types that contain UTF-16 and DBCS data in the same copybook, you can use the `USAGE NATIONAL` or `USAGE DISPLAY-1` qualifiers on individual fields.

**Note:** `DFHLS2WS`, `DFHLS2SC`, and `DFHLS2JS` do not support the `COBOL GROUP USAGE NATIONAL` clause.

## Querying XML from an application

You can write an application program to query a fragment of XML before transforming it into application data.

### About this task

If your application is going to process many different types of XML, you might want to query the XML to determine which `XMLTRANSFORM` resource to use to transform it to application data. This command might also be useful if your XML contains `<xsd:any>` elements.

### Procedure

1. In your application program, use the **TRANSFORM XMLTODATA** API command to query the XML:

```
EXEC CICS TRANSFORM XMLTODATA
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

You must specify the name of the channel and the name of the container that holds the XML. You do not have to specify an `XMLTRANSFORM` resource to query the XML. The above example queries the name of the first XML element and the length of the XML element. You can also query the type of the first XML element, the length of the type, and the namespace of the type.

2. Optional: If the application requires the namespace of the XML element, provide a data-area in which CICS can write the `ELEMNS` value.
3. Optional: After querying the XML, you can write application logic to determine which `XMLTRANSFORM` resource to use to transform the XML to application data.
4. Install the application program in CICS.

## Results

CICS reads the specified container and returns the information about the XML element to the application program.

## Handling XML by data type

If the XML schema contains global data types, one or more of which are referenced in the XML, you can generate metadata to support these global data types and then parse or transform the XML in an application program.

### Before you begin

To generate the correct metadata, you must run DFHSC2LS specifying the TYPES=ALL parameter.

### Procedure

1. To transform application data to XML by data type, use the **TRANSFORM DATATOXML** command:

```
EXEC CICS TRANSFORM DATATOXML
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
DATCONTAINER('SourceContainerName')
XMLCONTAINER('TargetContainerName')
ELEMNAME(elementName)
ELEMNAMELEN(elementNameLength)
ELEMNS(elementNamespace)
ELEMNSLEN(elementNamespaceLength)
TYPENAME(typeName)
TYPENAMELEN(typeNameLen)
TYPENS(typeNamespace)
TYPENSLEN(typeNamespaceLen)
```

*MyXmlTransformName* is the 32-character name of the XMLTRANSFORM resource that specifies the XML binding and schema; *MyChannelName* is the 16-character name of the channel that has the input and output containers; *SourceContainerName* is the 16-character name of the input container that holds the application data and *TargetContainerName* is the 16-character name of the output container that CICS populates with XML. You must also specify the XML element name, the namespace, the data type, and the namespace of the data type for the transformation.

2. To parse the XML by data type, use the **TRANSFORM XMLTODATA** command.

The options that you specify on the command depend on whether the XML has an `xmlns:type` attribute.

- If the XML uses the `xmlns:type` attribute, specify the following command in your application program:

```
EXEC CICS TRANSFORM XMLTODATA
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
DATCONTAINER('TargetContainerName')
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
TYPENAME(typeName) TYPENAMELEN(typeNameLength)
```

If the application also requires the namespace of the data type, add the TYPENS option to the API command. CICS returns the data type from the `xmlns:type` attribute in TYPENAME.

- If the XML does not use the `xmlns:type` attribute, the application program can specify the local name and namespace of the global data type. Use the following command in your application program:

```
EXEC CICS TRANSFORM XMLTODATA
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
DATCONTAINER('TargetContainerName')
TYPENAME(typeName) TYPENAMELEN(typeNameLength)
TYPENS(TypeNamespace) TYPENSLEN(typeNamespaceLen)
```

The application program must specify the type name and namespace but not the name and namespace. This command indicates that the application tells CICS which data type to use. When CICS is generating XML, CICS always adds the `xsi:type` attribute.

3. Install the application program in CICS.

### What to do next

Test that the application program generates and parses the XML as expected.

## Handling `<xsd:any>` data types

---

If you are working with an XML schema that contains one or more `<xsd:any>` data types, the XML assistants can map the data type to a pair of CICS containers. You can write an application program to parse the XML in the containers.

### Before you begin

You must map the XML schema using DFHSC2LS or DFHWS2LS using a mapping level of 2.1 or higher.

### About this task

When CICS transforms the data to XML, it puts the XML associated with the `<xsd:any>` data type in the first container and the namespace prefix declarations that are in scope in the second container.

### Procedure

1. To parse the XML data, use the **TRANSFORM XMLTODATA** command in your application program:

```
EXEC CICS TRANSFORM XMLTODATA
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
DATCONTAINER('TargetContainerName')
NSCONTAINER('NamespacesContainerName')
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

*MyXmlTransformName* is the 32-character name of the XMLTRANSFORM resource that specifies the XML binding and schema; *MyChannelName* is the 16-character name of the channel that has the input and output containers; *SourceContainerName* is the 16-character name of the input container that holds the XML and *TargetContainerName* is the 16-character name of the output container that CICS populates with application data; *NamespacesContainerName* is the 16-character name of the container that CICS populates with the namespace prefix declarations. Provide initial values for the ELEMNAME and ELEMNAMELEN options.

CICS returns the XML element and its length in the ELEMNAME and ELEMNAMELEN options.

2. Install the application program in CICS.

### What to do next

Test that the application program parses the XML correctly.

## Handling 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.

## Example

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

```

## Generating mappings from language structures

To create XML from application data or vice versa, you create the mappings to describe how CICS will transform the data and XML at run time. You can start from any application data record; for example, you can start with a `COMMAREA`, `VSAM` file, temporary storage queue, or a `Db2` record.

### Before you begin

Before you create the mappings, you must make sure that these preconditions have been completed:

- You must have a language structure that describes the application record in a partitioned data set. The language structure can be written in any of the high-level languages supported by the CICS XML assistant: `COBOL`, `PL/I`, `C`, and `C++`. If this mapping is for use with an Atom feed, and you are using any of the fields in your application record to provide metadata for the Atom entries (such as the name of an author), make sure that those fields are not nested in your language structure. You can have structures of nested fields within a field that provides the content for an Atom entry.
- You must configure the user ID under which `DFHLS2SC` runs to use `z/OS UNIX`.
- The user ID must have read permission to access the language structure and write permission to put the output in the appropriate directories on `z/OS UNIX`.
- 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, `DFHLS2SC` uses the Java version specified in the **JAVADIR** parameter.

### About this task

Use the CICS XML assistant to create the data mappings for the application record. For each high-level language supported by the CICS XML assistant, a few types of data are restricted or not supported. The CICS XML assistant issues error messages about any unsupported items that it identifies in your language structure. The reference information for the CICS XML assistant lists the restrictions that apply to each high-level language.

## Procedure

### 1. Run the DFHLS2SC batch job.

DFHLS2SC has optional parameters that you select to meet your requirements, such as selecting a particular code page or a name space. Use the following parameters as a minimum:

- a) Specify the high-level language of your language structure in the **LANG** parameter.
- b) If you are deploying the data mappings in a bundle, specify the name of a bundle resource in the **BUNDLE** parameter.  
If you are creating an XML binding for an Atom feed, do not specify this parameter.
- c) Specify the mapping level in the **MAPPING-LEVEL** parameter.  
If you are creating an XML binding for an Atom feed, you must use a mapping level of 3.0 or higher.  
For other situations, although you can use any mapping level, to get the most advanced mapping options use the latest mapping level.
- d) Optional: If you are creating an XML binding for an Atom feed, and your application data record contains timestamps in CICS ABSTIME format, specify the optional parameter **DATETIME=PACKED15** to map these fields as timestamps.
- e) Specify the location and the code page of the language structures that describe the application record in the **PDSMEM** and **PDSCP** parameters.
- f) Specify the name and location of the schema file in the **SCHEMA** parameter.  
The file extension is `.xsd`. If you are creating a bundle, do not specify a location. DFHLS2SC creates the XML schema, but not the directory structure, if the file does not already exist.
- g) Specify the name and location of the XML binding in the **XSDBIND** parameter.  
The file extension is `.xsdbind`. If you are creating a bundle, do not specify a location. DFHLS2SC creates the XML binding, but not the directory structure, if the file does not already exist.

**Tip:** Put the XML binding and the schema in the same directory structure to enable validation. Validation can be useful when you are testing out your application in a development or test environment. If you are creating a bundle, CICS puts the files in the same directory for you.

If you specify the **BUNDLE** parameter, the batch job creates a bundle directory structure on z/OS UNIX. The bundle directory has a META-INF subdirectory that contains the bundle manifest. The batch job also creates an XML schema and an XML binding in the bundle directory, using the file names that you specified for the **SCHEMA** and **XSDBIND** parameters. If you do not specify the **BUNDLE** parameter, the batch job creates the XML schema and XML binding only in the specified location.

### 2. Install the BUNDLE resource, or an ATOMSERVICE resource that specifies this XML binding.

The BUNDLE and ATOMSERVICE resources dynamically create an XMLTRANSFORM resource, which defines the location of the XML schema and binding file.

## Results

When you generate mappings from language structures, only one XML transformation is possible.

## Example

The following example shows DFHLS2SC with the minimum set of parameters specified.

```
//LS2SC JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHLS2SC,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB>//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
```

```
SCHEMA=example.xsd
/*
```

## What to do next

Write an application program to transform the application data to XML and vice versa. You can use the same mappings for both transformations. If you have created an XML binding for an Atom feed, continue with the steps to set up your Atom feed.

## Generating mappings from an XML schema

---

To create application data from XML, or XML from application data, that conforms to an existing XML schema, you create the mappings to describe how CICS will transform the data at run time. You can start from an XML schema or a WSDL document.

### Before you begin

You must have a valid XML schema or WSDL document. Before you create the mappings, you must make sure that these preconditions have been completed:

- You must have a valid XML schema or WSDL document.
- You must configure the user ID under which DFHSC2LS runs to use UNIX System Services.
- The user ID must have read permission to access the XML schema or WSDL document and write permission to put the output in the appropriate directories on z/OS UNIX.
- The user ID must have a sufficiently large storage allocation 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

Use the CICS XML assistant to create the data mappings for the XML schema.

### Procedure

1. Run the DFHSC2LS batch job.

DFHSC2LS has optional parameters that you can select to meet your requirements, such as selecting a particular code page or specifying how to handle variable-length character data. Use the following parameters as a minimum:

- a) Specify the location of your input file in the **WSDL** or **SCHEMA** parameter.  
You can either use a WSDL document or an XML schema. If your input file contains references to other schemas or documents on the Internet and the system uses a proxy server, specify the domain name or IP address and the port number of the proxy server.
- b) Specify the high-level language that you want to generate in the **LANG** parameter.  
The XML assistant supports COBOL, C, C++, and PL\I languages.
- c) If you are deploying the data mappings in a bundle, specify the name and location of a bundle in the **BUNDLE** parameter.

The XML assistant creates a library of supported transformations in the XML binding. For each global element in the input file, the assistant creates a separate transformation.

If you specify the **BUNDLE** parameter, the batch job creates a bundle directory structure on z/OS UNIX . The bundle directory has a META-INF subdirectory that contains the bundle manifest. The batch job also creates an XML binding in the bundle directory and places the language structures in the specified location. The XML assistant also places a copy of the input file in the bundle directory. If you do not specify the **BUNDLE** parameter, the batch job creates the language structures and XML binding only in the specified location.

2. Install the BUNDLE resource.

The BUNDLE resource dynamically creates an XMLTRANSFORM resource, which defines the location of the XML schema or WSDL document, the XML binding, and the language structures.

## Results

When you generate mappings from an XML schema, CICS generates a language structure for every global element that is present in the schema.

## Example

The following example shows DFHSC2LS with the minimum set of parameters specified.

```
//SC2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHSC2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB>//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*
```

## What to do next

Write an application program to transform the application data to XML, or XML to application data. You can use the same mappings for both transformations.

## Transforming application data to XML

---

You can write an application program to transform application data to XML.

### Before you begin

You must have an enabled XMLTRANSFORM resource that defines the XML binding and XML schema.

### About this task

The XML assistant generates the mappings in the XML binding. If you started with a language structure and used DFHLS2SC, only one transformation to the XML is possible. If you started with an XML schema, there might be many XML to language structure transformations so your application must select which XML element to generate.

### Procedure

1. The application program must create a channel and place the data associated with the language structure into a bit-mode container on that channel.
2. Use the **TRANSFORM DATATOXML** API command to transform the data to XML:

```
EXEC CICS TRANSFORM DATATOXML
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
DATCONTAINER('SourceContainerName')
XMLCONTAINER('TargetContainerName')
```

If the XMLTRANSFORM resource supports only a single transformation from the language structure, you do not have to specify the conversion type on the command. If many transformations are possible, add the following options to your application program:

```
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

These additional options indicate the XML element into which the application data is transformed and placed in the output container.

3. Install the application program.

## Results

When the application runs the **TRANSFORM DATATOXML** command, CICS checks the XMLTRANSFORM resource to find the mappings in the XML binding and transforms the application binary data to XML using the containers on the channel. The XML is placed in the container that the application specified on the XMLCONTAINER option. The XML conforms to the XML schema that is defined in the XMLTRANSFORM resource.

## What to do next

You can also use the same mappings to transform XML to application data. For details, see [“Transforming XML to application data” on page 392](#).

## Transforming XML to application data

---

You can write an application program to transform XML to application data. You can also query the XML before transforming it.

### Before you begin

You must have an enabled XMLTRANSFORM resource that defines the XML binding and XML schema.

### About this task

The CICS XML assistant generates the mappings in the XML binding. If you started with a language structure and used DFHLS2SC, only one transformation from the XML is possible. If you started with an XML schema, there might be many XML to language structure transformations so your application must select which language structure to use as output.

### Procedure

1. Create a channel and place the XML into a text mode container on that channel.  
If the application puts the XML in a BIT mode container, CICS tries to determine the encoding of the text data, but it can take longer to process the container and the encoding might not be correct.
2. Use the **TRANSFORM XMLTODATA** API command.  
If only one transform is possible for the XML, you can use the following command:

```
EXEC CICS TRANSFORM XMLTODATA
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
DATCONTAINER('TargetContainerName')
```

If more than one transformation is possible, use the following additional options:

```
EXEC CICS TRANSFORM XMLTODATA
XMLTRANSFORM('MyXmlTransformName')
CHANNEL('MyChannelName')
XMLCONTAINER('SourceContainerName')
DATCONTAINER('TargetContainerName')
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

In the second example, CICS returns the name of the element that was found in the ELEMNAME option. The application can then use the name of the element to determine which of the library of language structures to use to interpret the contents of the target container.

3. Install the application program.

## Results

When the application runs the **TRANSFORM XMLTODATA** command, CICS uses the details in the XMLTRANSFORM resource to transform the XML to application binary data using the containers on the channel.

## What to do next

You can also use the same mappings to transform application data to XML. For details, see [“Transforming application data to XML”](#) on page 391.

## Validating XML transformations

---

When you use the CICS XML assistant to map application data to XML, you can specify that the transformations that take place at run time are validated to ensure that they conform to the schema that is contained in the XML binding. You can perform validation on the transformation of XML to binary data or binary data to XML.

### Before you begin

During development and testing of your CICS application, full validation assists in detecting problems in the XML. However, full validation of the XML carries a substantial overhead and it is inadvisable to validate XML in a fully tested production application.

CICS uses a Java program to validate the XML against the schema. Therefore, you must have Java support enabled in your CICS region to perform validation.

### Procedure

1. Set up a JVM server in the CICS region.

The XML validator class can run 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.

- a) Install the sample JVM server DFHJVMS in group DFH\$OSGI or create your own JVM server.

For more information, see [Setting up a JVM server](#).

- b) If you created your own JVM server, 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.

2. Ensure that the XML binding and the schema are in the same location on z/OS UNIX.

The XMLTRANSFORM resource defines these files to CICS. You can use the **INQUIRE XMLTRANSFORM** command to check the location of each file.

3. Turn validation on for the application.

In CICS Explorer<sup>®</sup>, open the XMLTRANSFORM resource and edit the Validation Status field in the list of attributes. Alternatively, you can use CEMT or the SPI.

### Results

Check the system log to find out if the XML transformation is valid. Message DFHML0508 indicates that the XML was successfully validated and message DFHML0507 indicates that the validation failed.

## What to do next

When you no longer require XML validation for the application, update the XMLTRANSFORM resource to turn it off.



## Chapter 6. Security for SOAP web services

With the support provided by CICS Transaction Server for z/OS, you can either secure your SOAP messages with *transport security* (used for either SOAP or JSON messages) or *SOAP message security* (for SOAP messages only). For information about the architecture of SOAP web services, see [CICS and SOAP web services](#).

### Transport-based security

Transport-based security relies on technologies that are available as part of the TCP/IP and HTTP protocols. For example, TLS and HTTP basic authentication can be used to secure CICS web services. For information on TCP/IP and HTTP security options, see [Security for TCP/IP clients](#) and [Security for CICS web support](#).

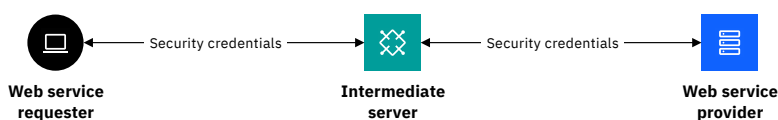


Figure 31. Transport-based security for SOAP messages

### SOAP message security

When you use transport-based security (as illustrated in [Figure 31 on page 395](#)), if the service requester identifies itself to an intermediate gateway, then the intermediate gateway identifies itself to the service provider, the target service normally runs with the identity of the intermediate gateway rather than the service requester. The Web Services Security (WSS): SOAP Message Security 1.0 specification (*WS-Security*) addresses this problem by allowing security credentials to be passed within the SOAP message itself, so that the credentials of the service requester can be passed through an intermediate gateway, and can still be used to identify the requester to the service provider.

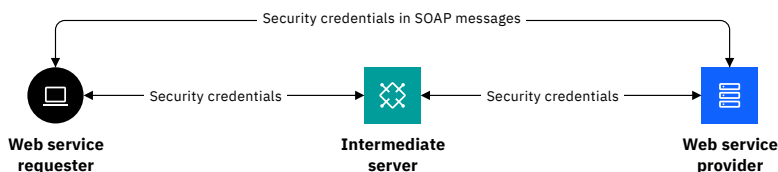


Figure 32. SOAP message security

For more information about SOAP message security, see [How it works: SOAP message security](#).

You can use a combination of transport-based security and SOAP message security to secure CICS web services. To decide which type, or which combination of capabilities, is best for you, see [Designing security for CICS web service providers](#) and [Designing security for CICS web service requesters](#).

## How it works: SOAP message security

CICS security for SOAP messages is based on support for standard web specifications, including the WS-Security and WS-Trust specifications. This support gives you various options for authentication, integrity, and confidentiality of SOAP messages. It is useful to understand the components that provide the support.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

6.3 Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

6.1 6.2 Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

- [“Support for WS-Security, WS-Trust, and WS-Policy specifications” on page 396](#)
- [“What SOAP message security offers” on page 396](#)
- [“Components of SOAP message security” on page 397](#)

## Support for WS-Security, WS-Trust, and WS-Policy specifications

The Web Service Security (*WS-Security*) specification provides a foundational set of SOAP message extensions for building secure web services by defining elements to be used in the SOAP header for message-level security. It specifies the use of security tokens, digital signatures, and XML encryption to protect and authenticate SOAP messages. It specifies the use of digital signatures to provide integrity for XML elements in a SOAP message, and it specifies the use of encryption to provide confidentiality for XML elements in a SOAP message. The specification allows you to protect the body of the message or any XML elements within the body or the header. You can give different levels of protection to different elements within the SOAP message. For more information, see the [Web Services Security: SOAP Message Security 1.0 specification](#).

The *Web Services Trust Language (WS-Trust)* specification extends WS-Security 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 by using a trusted third party. This third party is called a *Security Token Service (STS)*. For more information about WS-Trust, see the [Web Services Trust Language specification](#).

CICS does not support WS-Policy. Security constraints can be specified by using WS-Policy within the WSDL but these specifications are ignored by CICS. CICS uses the pipeline configuration file to specify the security handler. For more information, see [Configuring the pipeline for Web Services Security](#). CICS supports only the WS-Security 1.0 and WS-Trust 1.2 specifications.

## What SOAP message security offers

[Figure 33 on page 397](#) shows how a SOAP message can be extended with security data that is used to authenticate the service requester and to protect the message as it passes between the requester and a CICS web service provider. You configure SOAP message security either by using a [CICS security handler](#) or by using the [Trust client interface](#). The CICS security handler and the WS-Sec message header are both using WS-Security.

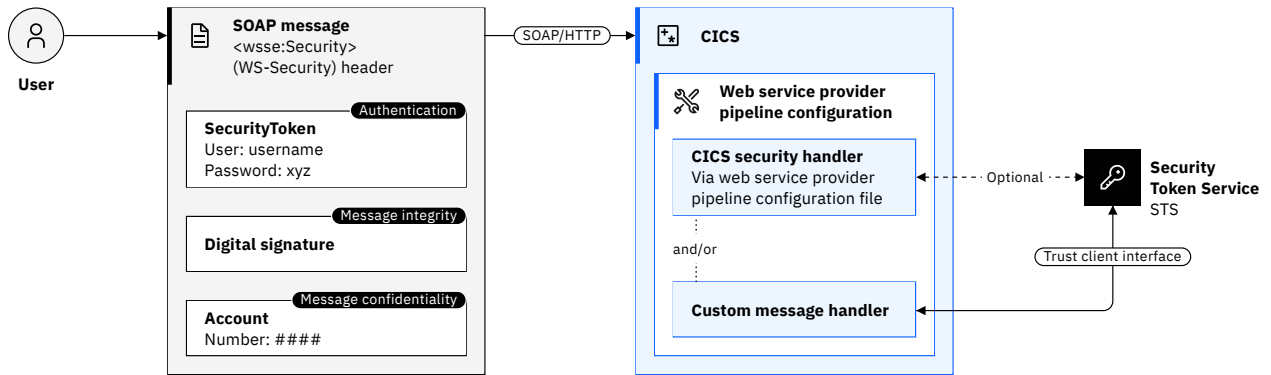


Figure 33. How CICS supports SOAP message security

SOAP message security in CICS offers you the following options for authentication and for message protection:

- Authentication mechanisms for deriving a user ID from an inbound message (CICS web service provider) or attaching a security token to an outbound message (CICS web service requester), including:
  - Basic authentication (for web service provider only)
  - X.509 certificate
  - ICRX identity token (web service provider only)
  - Identity assertion
  - Operation with a trusted third party (Security Token Service)

For more information, see [“How it works: Authentication for CICS with SOAP message security”](#) on page 399.

- For message integrity, signature validation of inbound message signatures and signature generation for the SOAP body on outbound messages. For more information, see [“How it works: Signing SOAP messages ”](#) on page 401 .
- For message confidentiality, decryption of encrypted data in inbound messages and encryption of the SOAP body content on outbound messages. For more information, see [“How it works: SOAP message encryption ”](#) on page 402.
- You can choose to both sign and encrypt a SOAP message to provide both message integrity and confidentiality. CICS always signs the SOAP message body first and then encrypts it.

**Note:** Web Services Security is potentially not conformant with SP800-131A. Web Services Security is configured by adding a security handler into the pipeline and CICS has no control over the processing in a user-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 triple DES encryption algorithm, which can be used to encrypt a SOAP body, is also nonconformant.

## Components of SOAP message security

### CICS security handler

To implement WS-Security in CICS for either a web service provider or a web service requester, you configure a security handler in the pipeline configuration file. CICS provides a security handler to cover the most common scenarios. For more information about using the CICS security handler, see [Configuring the pipeline for Web Services Security](#).

To use your own security procedures and processing, you can write a custom security handler to process secure SOAP messages. For information about writing your own security handler, see [Writing a custom security handler](#).

In addition to the CICS security handler, CICS provides a Trust client interface so that you can interact with a Security Token Service (STS) without using the CICS security handler. For information about using this interface, see [Invoking the Trust client from a message handler](#).

## Security Token Service (STS)

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, allowing web services from different trust domains to communicate successfully. Like a certificate authority in a TLS handshake, the STS guarantees that the service requester and the service provider can trust the credentials that are provided in the message. The trust is represented through the exchange of security tokens. An STS can issue, exchange, and validate these security tokens, and, in doing so, establish trust relationships between different trust domains.

The STS enables CICS to accept and send messages that have security tokens in the message header that are not directly supported by the CICS security handler; for example, LTPA and Kerberos tokens. CICS can be configured as a Security Token Service for SAML assertions. For information about using SAML assertions, see [Configuring CICS for SAML](#).

CICS acts as a Trust client. The CICS security handler uses the information in the pipeline configuration file to send a web service request to the STS.

In a service provider pipeline, depending on how you configure the security handler, the request that is sent to the STS can be used for either of the two things:

- To validate a security token in the WS-Security message header of the inbound message.
- To exchange a security token in the WS-Security message header for a security token that CICS can understand.

In a service requester pipeline, the request can be only one thing: to exchange a security token for a different type of token. The pipeline configuration file defines what type of token the STS issues to the security handler.

The security handler dynamically creates a pipeline to send the web service requests 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 through the pipeline to the web service requester.

For more information about using an STS, see [Configuring the pipeline for Web Services Security](#).

## Trust client interface

The Trust client interface enables you to interact with a Security Token Service (STS) directly, rather than using the CICS security handler. Using an STS offers more advanced processing of tokens than the processing that the CICS security handler offers: for example, you can enable specific processing to handle many tokens in the inbound message headers or to exchange multiple types of token for outbound messages. Using this interface, you can create a custom message handler to send your own web service request to the STS.

You can use the Trust client interface without enabling the CICS security handler in your service provider and service requester pipelines, or you can use the Trust client interface in addition to the CICS 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 is not deployed by using the CICS web services assistant. But it can also act as the Trust client interface to an 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, described in [Security containers](#). Using these containers, you 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.

For more information about interacting directly with an STS, see [Invoking the Trust client from a message handler](#).

## How it works: Authentication for CICS with SOAP message security

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

WS-Security is applied to the message by inserting a SOAP security header. WS-Security defines a vocabulary that can be used inside the SOAP envelope. The XML element `<wsse:Security>` is the container for security-related information. (`wsse` stands for web services security extension. `wsse` is an arbitrary prefix. Another prefix can be used, if it matches the namespace definition.)

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

The simplest form of security token is the username token, which is used to provide a username and (optional) password. A signed security token is one that is digitally signed by a specific authority. For example, an X.509 certificate is a signed security token. Security token usage for WS-Security is defined in separate profiles such as the username token profile and the X.509 token profile.

You can choose from the following authentication options for SOAP message authentication with CICS:

### Basic authentication (for web service provider only)

In service provider mode, CICS can accept a UsernameToken in the SOAP message header for authentication on inbound SOAP messages. The UsernameToken contains a Username element and a Password element. CICS verifies the Username and Password with RACF. If this check is successful, CICS places the Username in container DFHWS-USERID and processes the SOAP message in the pipeline. If CICS cannot verify the UsernameToken, it returns a SOAP fault message to the service requester.

Username tokens that contain passwords are not supported on outbound SOAP messages when CICS is the service requester. A custom security handler can be written to store and send passwords. Any such handler must be responsible for securing the credentials that are used. This pattern is strongly discouraged because it is inherently insecure.

### X.509 certificate authentication

6.1 6.2

An X.509 certificate that is used for signing SOAP messages can also be used for authentication. In service provider mode, CICS maps the certificate to a RACF user ID and places the user ID in the container DFHWS-USERID. In service requester mode, CICS can send an X.509 certificate in the SOAP message header to the service provider that is then used for authentication purposes. The certificate is identified with an element in the pipeline configuration file.

6.3 Beta

Support for X.509 certificate authentication for the WS-Security feature is removed as of CICS TS 6.3.

## Trusted authentication (identity assertion)

In service provider pipelines, CICS can accept a UsernameToken or X.509 certificate in the SOAP message header as trusted. This type of security token typically contains a username and password, but in this case, the password is not required. When an X.509 certificate is used, the certificate is mapped to a RACF user ID. CICS trusts the provided identity and places the user ID in container DFHWS-USERID.

This option is a form of identity assertion in which an already-authenticated user identity is forwarded to CICS, and CICS accepts the identity without having to re-authenticate.

In service requester pipelines, CICS can send a username token without the password in the SOAP message header to the service provider. The user ID placed in the identity token is the contents of the DFHWS-USERID container. CICS can also send an X.509 certificate. The certificate is identified with an element in the pipeline configuration file.

## ICRX-based identity propagation (for web service provider only)

An ICRX identity token is a z/OS identifier that binds a distributed ID to a (potentially shared) SAF user ID. User activities can be audited and tracked by using the identity by which they are known outside of z/OS, but authorized by using their SAF identity. In service provider mode, you can implement identity propagation with IBM DataPower Gateway acting as an intermediate server between web service requester applications and the CICS web service provider. The web service requester applications authenticate with IBM DataPower Gateway. IBM DataPower Gateway then creates a z/OS ICRX (*Extended Identity Context Reference*) identity token and forwards the token to CICS in a WS-Security header over the trusted TLS connection. The ICRX token allows CICS to map the distributed user ID to a RACF 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.

ICRX-based identity propagation allows user access to CICS resources to be audited and tracked through the distributed identity, while using the mapped RACF user ID for resource authorization at the same time.

For more information about using an ICRX identity token, see [Configuring provider mode web services for identity propagation](#).

## Advanced authentication (Security Token Service)

6.1 6.2

In web service provider and requester pipelines, you can verify or exchange security tokens with a Security Token Service (STS) for authentication purposes. The STS enables CICS to accept and send messages that have security tokens in the message header that are not directly supported by the CICS security handler; for example, LTPA and Kerberos tokens or SAML assertions. For more information, see [Security Token Service](#).

You either configure the CICS security handler to define how CICS interacts with an STS or write your own message handler to use the Trust client interface. Whichever method you select, use TLS to secure the connection between CICS and the STS.

For an inbound message, you can 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 only exchange a username token for a security token.

6.3 Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

6.3 Beta

Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

## How it works: Signing SOAP messages

Integrity is applied to a SOAP message to ensure that no one illegally modifies the message while it is in transit. Essentially, integrity is provided by generating an XML digital signature on the contents of the SOAP message. If the message data changes illegally, the signature would no longer be valid.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

Generating a digital signature involves encrypting a message digest with a private key to create the electronic equivalent of a handwritten signature. You can use a digital signature to verify the identity of the signer and to ensure that nothing altered the SOAP message since it was signed.

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.

The [WS-Security specification](#) allows the contents of the <Header> and the <Body> to be signed at the element level. That is, in a message, individual elements can be signed or not, or can be signed with different signatures or by using different algorithms. For example, in a SOAP message that is 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 CICS security message handler can verify the digital signature on individual elements in the SOAP <Header> and the <Body>. The CICS security handler verifies the following elements:

- Signed elements that it encounters in the <Header>
- Signed elements in the SOAP <Body>. If the handler is configured to expect a signed body, CICS rejects with a fault any SOAP message in which the body is not signed.

For outbound messages, the security message handler can sign the SOAP <Body> only; it does not sign the <Header>. The security handler's configuration information specifies the algorithm and key that is used to sign the body.

To use CICS XML digital signatures, z/OS Integrated Cryptographic Service Facility (ICSF) must be started and configured with cryptographic devices. For more information, see [z/OS Integrated Cryptographic Service Facility \(ICSF\)](#).

**Note:** The performance overhead of XML signatures is significant.

### Signature algorithms

CICS supports the signature algorithms that are required by the XML Signature specification. Each algorithm is identified by a universal resource identifier (URI).

Algorithm	URI
<b>Digital Signature Algorithm with Secure Hash Algorithm 1 (DSA with SHA1) Supported on inbound SOAP messages only.</b>	<a href="http://www.w3.org/2000/09/xmldsig#dsa-sha1">http://www.w3.org/2000/09/xmldsig#dsa-sha1</a>
<b>Rivest-Shamir-Adleman algorithm with Secure Hash Algorithm 1 (RSA with SHA1)</b>	<a href="http://www.w3.org/2000/09/xmldsig#rsa-sha1">http://www.w3.org/2000/09/xmldsig#rsa-sha1</a>



Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

6.3 Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

6.1 6.2 Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

The sender uses a secret key (also known as a *symmetric key*) to encrypt the message, and the receiver uses the same key to decrypt the message.

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.

The WS-Security specification allows some of the contents of the <Header> and all of the contents of the <Body> to be encrypted at the element level. That is, in a message, individual elements can have different levels of encryption, or can be encrypted by using different algorithms. For example, in a message that is 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 be safely encrypted by using a less secure (but faster) algorithm and other information might safely remain unencrypted.

CICS supports encryption of the SOAP message body by using a symmetric algorithm such as the Triple DES algorithm or the AES algorithm. A symmetric algorithm is where the same key is used to encrypt and decrypt the data. This key is known as a symmetric key. The symmetric key is included in the message and is itself encrypted using the intended recipient's public key with the asymmetric key encryption algorithm RSA 1.5. This method gives you increased security because the asymmetric algorithm is complex and it is difficult to decrypt the symmetric key. However, you obtain better performance because most of the SOAP message is encrypted with the symmetric algorithm, which is faster to decrypt.

For inbound messages, the CICS 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 the following elements:

- Elements that it encounters in the <Header> in the order that the elements are found.
- Elements in the SOAP <Body>.

If the handler is configured to expect an encrypted body (by using the <expect\_encrypted\_body> element), CICS rejects with a fault any SOAP message in which the body is not encrypted.

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>. When the security message handler encrypts the <Body>, all elements in the body are encrypted with the same algorithm and use the same key. The algorithm and information about the key are specified in the handler's configuration information.

CICS supports the encryption algorithms that are required by the XML Encryption specification. Each algorithm is identified by a universal resource identifier (URI). For supported algorithms, see [Web Services Security: SOAP Message Security](#).

To use CICS XML encryption, z/OS Integrated Cryptographic Service Facility (ICSF) must be started and configured with cryptographic devices. For more information, see [z/OS Integrated Cryptographic Service Facility \(ICSF\)](#).

**Note:** The performance overhead of XML encryption is significant.

## Encryption algorithms

CICS supports the encryption algorithms that are 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)	<a href="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc">http://www.w3.org/2001/04/xmlenc#tripleDES-cbc</a>
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	<a href="http://www.w3.org/2001/04/xmlenc#aes128-cbc">http://www.w3.org/2001/04/xmlenc#aes128-cbc</a>
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	<a href="http://www.w3.org/2001/04/xmlenc#aes192-cbc">http://www.w3.org/2001/04/xmlenc#aes192-cbc</a>
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	<a href="http://www.w3.org/2001/04/xmlenc#aes256-cbc">http://www.w3.org/2001/04/xmlenc#aes256-cbc</a>

## Example of an encrypted message

This example shows a SOAP message that is 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">MIICChDCCAc2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHcEMMAoGA1UEChMD
SUJNMRMwEQYDVQDEwPXAwwsIF1hdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
MDELMAkGA1UEBhMCRC0lxDAAKBgNVBAoTAA0lCTTEtMBEGA1UEAxMKV21sbCBZYXRlcjCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwYkCgYEArsRj/n+3RN75+jaxuOMBWShvZCB0egv8qu2UwLWEieioGePsR
6Ku4SuHbBwJtWNr0xBTAAS91Ea70yhVdppxOnJB0CiERg7S0HUdP7a8JXPFZA+BqV63JqRgJyxN6
msfTAvEMR07LIXmZAt62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAa0BtTCBqjA/BglghkgBhvhC
AQ0EMhMwR2VuzXJhdGVkIGJ5IHRoZSBTZW51cm10eSBTZjJ2ZXIgzM9yIHovT1MgKFJBQ0YpMDGg
A1UdEQQxMC+BEVdZQVRFU0BVsy5JQk0uQ09NggdJQk0uQ09NhgTAV1cuSUJNLKNPtYcECRR1BjAO
BgNVHQ8BAf8EBAMCAfYwHQYDVR00BBYEFMiPX6VZKP5+mSOY1TLNQGvVJzu+MA0GCSqSqsIb3DQEB
BQUAA4GBAHdrS409Jhoe67pHL2gs7x4SpV/N0uJnn/w25sjjop3RLgJ2bKtK6RiEevhCDim6tnYW
NyjBL1VdN7u5M6ktfd+HutR/HnIrQ3qPkXZK4ipgC0RWDJ+8APLySCxtFL+J0LN9Eo6yjiHL68mQ
uZbTH2LvzFMy4PqEbmVKbmA87a1F

  </wsse:BinarySecurityToken>
  <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/> 2
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#x509cert00"
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/> 3
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>M6bDQ0tJrvX0pEjAEIcF6bq6MP3ySmB4TQ0a/B5U1Qj1vWjD56V+GRJbF7ZCES5oJwCJHRVKW1ZB5 4
      Mb+aUzSWlsoHzHQixc1JchgWcIyIn+E2TbG3R9m0zHD3XQsKTyVa0T1R7VP0MbD1ZLNDIomxjZn2
      p7JfxywXk0bcSLhdZnc=</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>kgqvKnMcgIUUn7r11vkFXF0g4SodEd3dxAJo/mVN6ef211B1MZelg70yjEHf4ZXw1Cdt0FebIdlnK 6
      rrrksq11Mpw6So7ID8zav+KPQUGm4+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** This is the algorithm that was used to encrypt the symmetric key.

**3** This references the binary security token that contains the public key that is used to encrypt the symmetric key.

- 4 This is the encrypted symmetric key that was used to encrypt the message.
- 5 This is the encryption algorithm that was used to encrypt the message.
- 6 This is the encrypted message.

## Designing security for CICS web service providers

Two primary scenarios are possible for a CICS web service provider: a direct request to a CICS web service provider or a request through an intermediate server that uses a third-party authentication server. To secure each of these scenarios, consider the implications for different aspects of security and decide which options are the best for you. Examples illustrate some recommended options.

For the equivalent considerations for web service requester applications, see [Designing security for CICS web service requesters](#).

- “[Scenarios for CICS web service providers](#)” on page 405
- “[Security design considerations for CICS web service providers](#)” on page 406

For more information, see [Configuring security for CICS web services](#).

### Scenarios for CICS web service providers

In [Figure 34 on page 405](#), the web service requester sends a request directly to a CICS web service provider application.

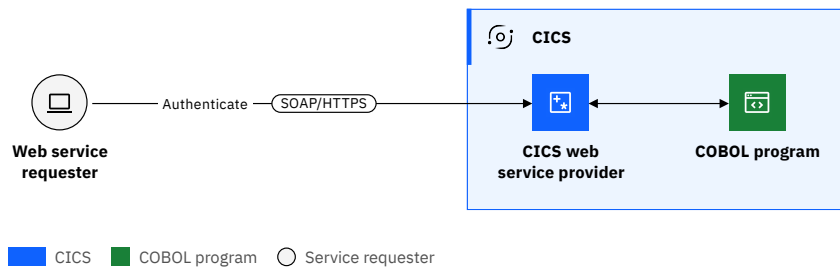


Figure 34. Direct request to CICS web service provider

In [Figure 35 on page 405](#), the web service requester calls an intermediate server, which authenticates the request with a third-party authentication server, then passes the identity of the client to the CICS web service provider.

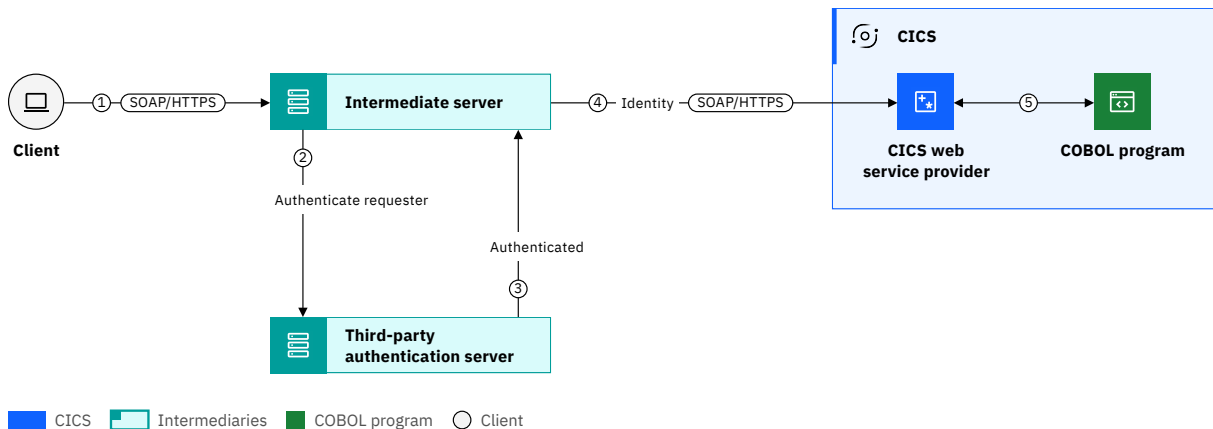


Figure 35. Request from an intermediate server

## Security design considerations for CICS web service providers

When you design security for CICS web service provider scenarios, consider the implications for:

- [Authentication and identification](#)
- [Authorization](#)
- [Confidentiality and integrity](#)
- [Trust](#)
- [Audit](#)

These considerations are explored further in this document.

## Authentication and identification considerations for web service provider security

### Who authenticates the web service requester? Is it CICS or an intermediate server?

CICS can authenticate a web service requester directly, or an intermediary can provide an authentication service to CICS. In this case, the intermediate server authenticates the client and then passes the identity of the client to CICS across a trusted connection. The client's identity can be passed to CICS as an [asserted user ID](#) or a [distributed identity](#).

### Will you use transport-based or SOAP message-based authentication?

Transport security is recommended for point-to-point requests where no intermediate server is involved. If you use transport-based authentication, you can configure HTTP basic authentication or TLS client authentication.

Use basic authentication only with HTTPS because the credentials are sent as cleartext and are vulnerable to packet sniffing. TLS client authentication is more secure than basic authentication because a client certificate is signed.

Use SOAP message-based authentication (WS-Security) when security credentials need to flow in the SOAP message through a number of intermediaries. If you use SOAP message-based security, you can configure the CICS security handler to authenticate with one of the following types of token:

- Username tokens that contain only a username.
- Username tokens that contain a username and password.
- X.509 digital certificate
- ICRX token.

6.3 Beta

Support for X.509 certificate authentication for the WS-Security feature is removed as of CICS TS 6.3.

If other token types are used, you can configure the CICS security handler to call a [Security Token Service \(STS\)](#) or you can write a custom security handler to do your own, customized security processing.

**Recommended:** Use TLS to secure the connection to the STS.

For more information about the types of authentication token that can be used with CICS web services, see [How it works: Authentication for CICS with SOAP message security](#).

In third-party authentication scenarios, SOAP-message authentication is often used along with transport-based authentication. The client's identity is flowed in a SOAP security header and the intermediate server is authenticated by using TLS client authentication.

### Do you need to implement different authentication mechanisms for different services or different types of client?

Instead of having a general rule for the application, the type of authentication mechanism can be decided for specific services. It might be appropriate to run read-only services with a generic user ID, where more sensitive services might need the requester to authenticate. In CICS, this split can be made by running secured services on a different pipeline to unsecured services.

## Authorization considerations for web service provider security

### Will you configure the CICS task in the web service provider to run with the RACF user ID of the end user?

Configuring the CICS task in this way enables fine-grained authorization and auditing.

- Where no intermediaries are used, CICS authorization processing can be based on the RACF user ID that is used to authenticate, or a RACF user ID that is associated with the security token that is used to authenticate.
- Where intermediaries are used, CICS authorization processing can be based on the asserted RACF user ID or the distributed identity that is passed by the intermediary. CICS can resolve a distributed identity that is contained in an ICRX identity token to a RACF user ID.

Use [Surrogate security](#) to ensure that the intermediary has the correct authority to start work on behalf of the asserted identity.

### Do you want to limit access to CICS resources to a specific, technical user ID?

Instead of configuring the CICS task in the web service provider to run with the RACF user ID of the end users, you can specify a RACF user ID in a URIMAP for the pipeline alias transaction.

If the web service does not need to be secured, requests can be allowed to run under the CICS default user ID.

## Confidentiality and integrity considerations for web service provider security

6.1

6.2

### TLS, XML signatures and XML encryption, or non-encrypted?

TLS can be used for integrity and confidentiality for point-to-point connections by using either the TLS support in CICS or AT-TLS. However, TLS does not ensure end-to-end message integrity or encryption because the message is not protected within intermediate servers, as illustrated in [Figure 35 on page 405](#).

XML signatures and XML encryption can be used to ensure integrity and confidentiality end-to-end, including within intermediate servers. However, this capability is rarely used because the performance overhead of XML signature processing and XML encryption in CICS is significant. As an alternative, these capabilities are often implemented in an intermediate server such as IBM DataPower Gateway.

**Recommendation:** If you need XML signature processing or XML encryption, consider implementing these capabilities in an intermediate server, such as IBM DataPower Gateway, instead of directly in CICS.

If SOAP messages do not contain critical data, or if the messages are only transmitted within an internal secure network, then it is reasonable to flow requests over non-encrypted HTTP connections.

6.3

Beta

Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3. You can still use Transport Layer Security (TLS) to secure SOAP messages. As a result, the XML Toolkit for z/OS is no longer required.

## Trust considerations for web service provider security

### How do you ensure that the intermediate server is a trusted server?

TLS client authentication can be used so that the intermediate server provides a client certificate that is validated by CICS. Successful client authentication requires that the certificate authority (CA) that signed the client certificate is considered trusted by CICS. To be considered trusted, the certificate of the CA must be in the CICS key ring.

### How do you ensure that the authentication token is trusted?

An authentication token, for example a SAML token, can be signed. The signature is used to verify the issuer of the SAML token and to ensure the integrity of the token.

## Audit considerations for web service provider security

### Does the web service request need to be audited in CICS?

Typically, you need to capture the RACF user ID that is used for running the CICS task. This information is recorded in the CICS SMF in type 110, subtype 01 records. When identity propagation is used, the distributed identity from the ICRX is also recorded.

### Will you create an audit trail in CICS or in an intermediate server, or both?

You might need an audit trail in all the servers that are involved in processing a request. IBM DataPower Gateway is an ideal place to capture audit information because it provides a configurable solution that supports multiple format log records that can be stored on the appliance or transferred off-device.

Review these design examples that offer different configurations.

## Design example: Securing a direct request with TLS client authentication

This scenario shows how TLS client authentication is used to secure a web service request to a CICS web service provider application. In this example, the web service requester application connects directly to CICS.

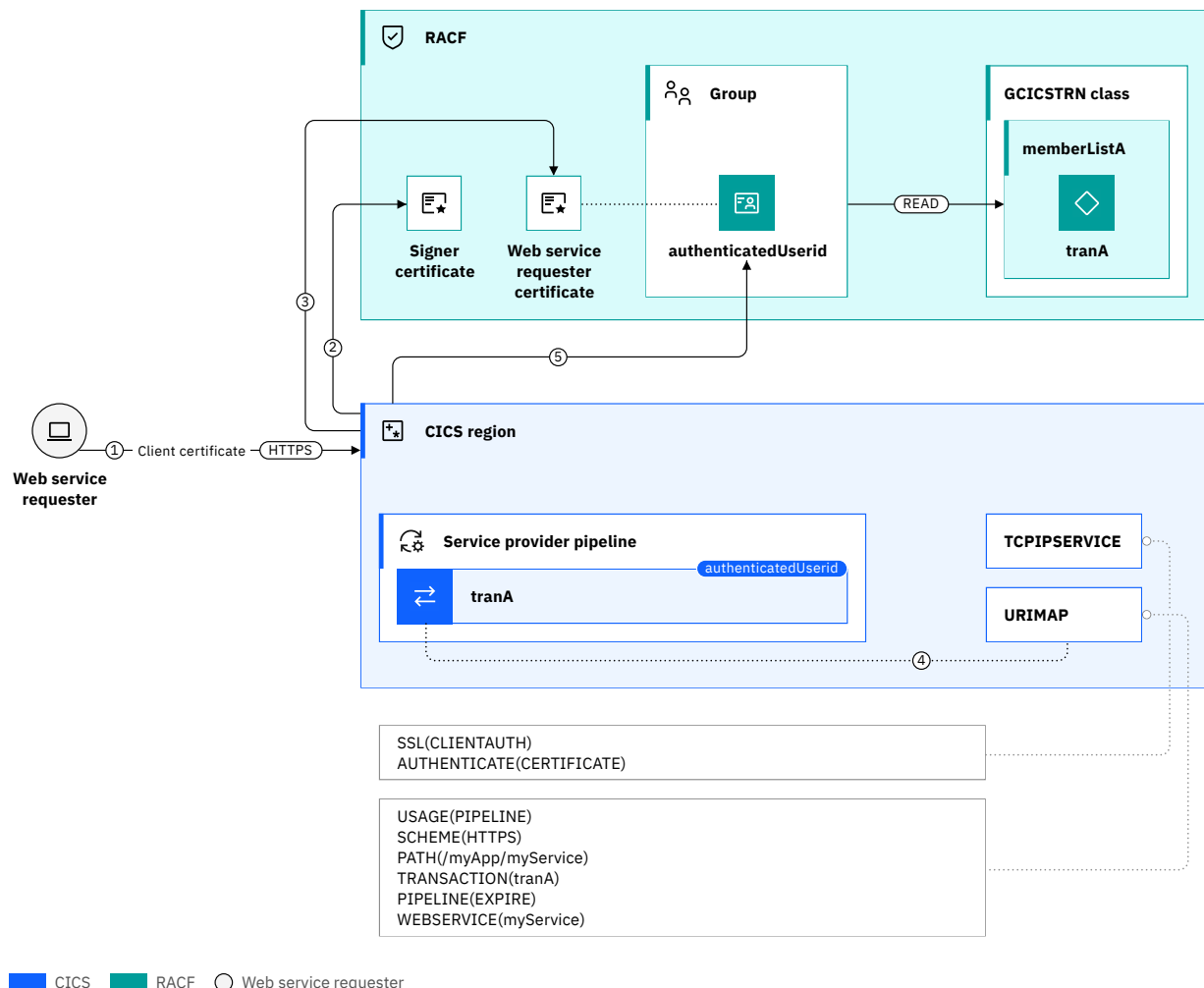


Figure 36. TLS authentication in a web service provider scenario

The TCPIPSERVICE resource in CICS specifies SSL(CLIENTAUTH), which requires the service requester to send a client certificate. It also specifies AUTHENTICATE(CERTIFICATE) so that CICS uses the client certificate to determine the authenticated user ID.

The URIMAP resource specifies the web service, pipeline, and transaction ID for the request. It also specifies SCHEME(HTTPS) to restrict access to the web service to HTTPS requests.

1. The web service requester application sends a SOAP/HTTPS request to CICS.
2. A TLS handshake occurs. CICS sends its server certificate and asks the service requester to provide its certificate. CICS validates the chain of trust by checking that the client certificate issuer (signer certificate) is in the RACF keyring.
3. CICS also maps the service requester certificate to a RACF user ID (the *authenticatedUserid*).
4. CICS finds the URIMAP that matches the request and determines the web service to start and the service provider pipeline to process the request. The URIMAP also determines the transaction ID *tranA* to run the request.
5. CICS queries RACF to verify that the *authenticatedUserid* is authorized to run transaction *tranA*. This validation is achieved by calling RACF to check that the *authenticatedUserid* belongs to a group that has READ authority to the member list (*memberListA*) containing *tranA*, in the GCICSTRN class.

When AT-TLS is used, the web attach transaction is not required in most situations. In this case, the pipeline alias transaction runs directly with the authenticated user ID.

## Configuration example: Securing a direct request with TLS client authentication

This task explains how to configure a CICS web service provider that supports authentication by using a TLS client certificate.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a direct request with TLS client authentication](#).

You must be familiar with web services and TLS.

You must complete these tasks:

- Completed the task [Configuring CICS to use SSL](#).
- Implemented a CICS web service provider as described in [Developing SOAP web services](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to accept web service requests over TLS. The CICS region identifies itself to the client by using the region's TLS server certificate. The client identifies itself to CICS by using its TLS client certificate that maps to a RACF user ID. This user ID is used to run the CICS web service attach transaction. The user ID is also used to run the web service when started through the client that sends the web service request.

This task assumes the following definitions:

- *keyRingA* is the name of the RACF key ring that is associated with the CICS region.
- *clientCA* is the RACF key ring label for the client certificate's signing CA.
- *clientCACertificateDataSet* is the data set containing the client certificate's signing CA.

- *clientCertificateDataSet* is the data set containing the client certificate (if a specific client certificate is mapped to a RACF user ID).
- *certificateLabel* is the RACF label for the client certificate in RACF.
- *groupA* is the RDO group name.
- *tcPIPService* is the name of a TCPIP SERVICE definition.
- *tcPIPPort* is a TCPIP port that is used by CICS to listen for the web service requests.
- *webServicePipe* is the name of a PIPELINE definition.
- *webServiceProvider* is the name of the webServiceProvider program.
- *soapProviderXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.
- *matchID* an identifier for the security request recording (SRR) report.
- *regionUseridA* is the region user ID for the CICS region.
- *clientUserid* is the user ID associated with the client certificate (from the requester) that runs the web attach transaction.
- CN=myClient.host.com, O=IBM, C=US is the client certificate subject's distinguished name value.

## Procedure

1. Import the client's CA certificate into the RACF key ring.

First, export the client's CA certificate (public key) from their keystore and transfer it to a sequential data set on z/OS. Now import the certificate into the CICS region's key ring by using the following RACF commands:

```
RACDCERT ID(regionUseridA) ADD('clientCACertificateDataSet') WITHLABEL('clientCA') TRUST
```

2. Identify the client's CA certificate as a trusted (CERTAUTH) certificate in the RACF keyring:

```
RACDCERT ID(regionUseridA) CONNECT(RING(keyRingA) LABEL('clientCA') USAGE(CERTAUTH))
```

3. The client's certificate must be mapped to a RACF user ID. Client certificates can be associated with a RACF user ID in two ways. Each expected client certificate can be mapped to a RACF user ID individually, or groups of certificates that match a specified pattern can be mapped to a RACF user ID. This latter approach significantly reduces the administration burden of managing digital certificates and is the approach that is described here.

Use the RACDCERT MAP command to define a certificate name filter. This command maps the certificate subject's distinguished name (DN) to a RACF user ID. Certificate name filtering supports generic filters allowing multiple certificates to be associated with a single RACF user ID.

- a. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

**Important:** In this command, the syntax of the SDNFILTER is significant. Use periods to separate the components of the distinguished name and remove any spaces between DN components.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the *z/OS® Security Server RACF Command Language Reference*.

- b. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

```
RACDCERT MAP ID(clientUserId) SDNFILTER('CN=myClient.host.com.0=IBM.C=US')
WITHLABEL('certificateLabel')
```

- c. Refresh the DIGTNMAP RACF class for the changes to take effect.

Enter the following RACF command:

```
SETROPTS RACLIST(DIGTNMAP) REFRESH
```

When this certificate is presented to the CICS region as part of a TLS handshake, the CICS region runs the subsequent transaction under the *clientUserId*, which is associated with the certificate in RACF.

4. Define the TCPIP SERVICE.

A TCPIP SERVICE is needed to listen for connections from the client. The connection uses TLS and requires the client to authenticate with a TLS certificate that is mapped to a user ID in RACF.

```
DEFINE TCPIP SERVICE(tcpipService) GROUP(groupA)
PORT(tcpipPort)
SSL(CLIENTAUTH) AUTHENTICATE(CERTIFICATE)
URM(NONE) PROTOCOL(HTTP) TRANSACTION(CWXN)
```

5. Define the PIPELINE.

The PIPELINE provides information about the message handler programs that act on a service request and on the response.

```
DEFINE PIPELINE(webServicePipe) GROUP(groupA)
CONFIGFILE(soapProviderXmlFile) SHELF(shelfDir)
WSDIR(wsProviderDir)
```

An example CONFIGFILE can be found in <cics-install-dir>/samples/pipelines/basicsoap11provider.xml.

6. Authorize *clientUserId* to the pipeline alias transaction CPIH and subsequent processing.

The following commands associate the *clientUserId* with the *taskUserGroup* and grant the group READ access to the PIPEUSER member list that contains CPIA, CPIH, CPIL, and CPIQ:

```
RDEFINE GCICSTRN PIPEUSER ADDMEM(CPIA,CPIH,CPIL,CPIQ)
UACC(NONE)
PERMIT GCICSTRN PIPEUSER ID(taskUserGroup) ACCESS(READ)
CONNECT clientUserId GROUP(taskUserGroup)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

If the PIPEUSER member list is already defined and associated with a user group, then you need only to add *clientUserId* to that user group.

7. Install the definitions in group *groupA*. Installing the PIPELINE creates URIMAP and WEBSERVICE resources for the web service.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and a URIMAP resource:

```
DFHS00107 02/07/2022 14:27:00 APPLID TCPIP SERVICE tcpipService has been opened on port tcpipPort
at IP address ANY.
DFHPI0701 I 02/07/2022 14:27:00 APPLID regionUseridA PIPELINE webServicePipe has been
created.
DFHPI0204 I 02/07/2022 14:27:00 APPLID regionUseridA PIPELINE webServicePipe is now ENABLED and
is ready for use.
DFHPI0703 I 02/07/2022 14:27:00 APPLID regionUseridA PIPELINE webServicePipe is about to scan the
WSDIR directory.
DFHPI0901 I 02/07/2022 14:27:00 APPLID regionUseridA New WEBSERVICE webServiceProvider is being
created during a scan against PIPELINE webServicePipe.
DFHPI0910 I 02/07/2022 14:27:00 APPLID regionUseridA WEBSERVICE webServiceProvider within
PIPELINE webServicePipe has been created.
```

```
DFHPI0915 I 02/07/2022 14:27:00 APPLID regionUserida WEBSERVICE webServiceProvider is now
INSERVICE and is ready for use.
DFHPI0903 I 02/07/2022 14:27:00 APPLID regionUserida New URIMAP £055300 is being created during a
scan against PIPELINE webServicePipe for
WEBSERVICE webServiceProvider.
DFHWB1560 02/07/2022 13:47:00 APPLID regionUserida URIMAP £055300 has been created.
DFHPI0704 I 02/07/2022 14:27:00 APPLID regionUserida PIPELINE webServicePipe Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The TCPIPSERVICE must have an OPENSTATUS of OPEN.
- The PIPELINE must have an ENABLESTATUS of ENABLED.
- The WEBSERVICE must have a STATE of INSERVICE.
- You must have a generated URIMAP with an ENABLESTATUS of ENABLED.

To validate the security environment is functioning correctly when you run this example, you can use the CICS security request recording (SRR) utility. The SRR logs all authorization calls and the matching response codes.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **Web** tab and set the **TCP/IP Service** field to *tcpipService*. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

The web service client can now start the CICS web service provider.

## Design example: Asserting an identity to the CICS web service provider

In this scenario, a client authenticates with an intermediate server, which then asserts the client's identity to the CICS web service provider.

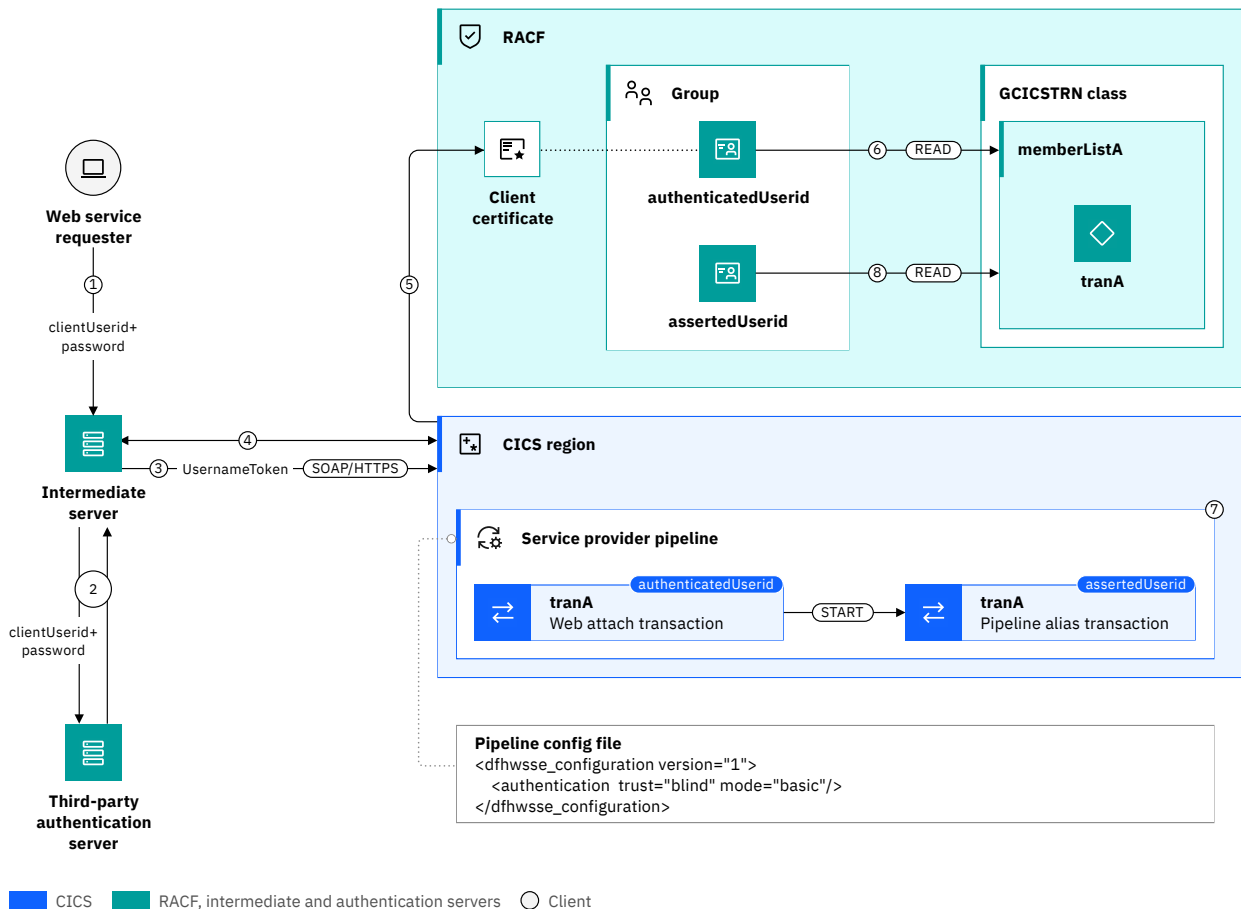


Figure 37. Identity assertion in a web service provider scenario

1. The client sends a request to the web service requester application.
2. The web service requester application authenticates the client with a third-party authentication server.
3. The web service requester application sends a SOAP/HTTPS request to CICS containing a *UsernameToken* in a SOAP header. The *UsernameToken* contains the RACF ID of the client. Note that this scenario requires that the client authenticates with a RACF ID, or the third-party authentication server maps the client's distributed identity to a RACF user ID.
4. CICS responds with its TLS server certificate and asks for the client's certificate. The web service requester application then validates the CICS server certificate and sends its TLS client certificate. The request is sent over HTTPS. Therefore, encryption and integrity are enabled at the transport level.
5. CICS validates the client certificate and maps the client certificate to a RACF user ID (*authenticatedUserid*).
6. The web attach transaction runs with the user ID mapped from TLS client certificate (*authenticatedUserid*).
7. The web service provider pipeline includes the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic. CICS puts the asserted user ID in the DFHWS-USERID container.

8. The pipeline alias transaction runs under the asserted user ID (*assertedUserid*) and all further resource security checks for this request (for example, program and database security) are then performed against the asserted user ID.

Use TLS client authentication to establish trust for an identity assertion scenario like the one described here. Additional trust can be configured by using [surrogate security](#), where the user ID mapped from TLS client certificate must have the correct authority to start work on behalf of the asserted identity.

## Configuration example: Asserting an identity to the CICS web service provider

This task explains how to configure a CICS web service provider that supports authentication by using a TLS client certificate, and then identity assertion by using blind trust.

### Before you begin

This configuration task is based on the example security scenario [Configuration example: Asserting an identity to the CICS web service provider](#). You must be familiar with web services and TLS.

Before you begin this task, you must complete these tasks:

- Complete the task [Configuring CICS to use SSL](#).
- Implement a CICS web service provider as described in [Developing SOAP web services](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to accept web service requests over TLS. The CICS region identifies itself to the client by using the region's TLS server certificate. The client identifies itself to CICS by using its TLS client certificate. The certificate maps to a RACF user ID and this user ID is used to run the CICS web service attach transaction. The client then sends the web service request that asserts the user ID that is used to run the requested web service.

This task assumes the following definitions:

- *keyRingA* is the name of the RACF key ring that is associated with the CICS region.
- *clientCA* is the RACF key ring label for the client certificate's signing CA.
- *clientCACertificateDataSet* is the data set containing the client certificate's signing CA.
- *clientCertificateDataSet* is the data set containing the client certificate (if a specific client certificate is mapped to a RACF user ID).
- *CN=myClient.host.com, O=IBM, C=US* is the client certificate subject's distinguished name value.
- *certificateLabel* is the RACF label for the client certificate in RACF.
- *groupA* is the RDO group name.
- *tcpipService* is the name of a TCPIP SERVICE definition.
- *tcpipPort* is a TCPIP port that is used by CICS to listen for the web service requests.
- *webServicePipe* is the name of a PIPELINE definition.
- *webServiceProvider* is the name of the webServiceProvider program.
- *soapProviderXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.

- *uriMapA* is the generated name of the URIMAP that is created by CICS when the PIPELINE is installed.
- *matchID* an identifier for the security request recording (SRR) report

## Procedure

1. Import the client's CA certificate into the RACF key ring.

First, export the client's CA certificate (public key) from their keystore and transfer it to a sequential data set on z/OS. Now import the certificate into the CICS region's key ring by using the following RACF commands:

```
RACDCERT ID(regionUseridA) ADD('clientCACertificateDataSet') WITHLABEL('clientCA') TRUST
```

2. Identify the client's CA certificate as a trusted (CERTAUTH) certificate in the RACF keyring:

```
RACDCERT ID(regionUseridA) CONNECT(RING(keyRingA) LABEL('clientCA') USAGE(CERTAUTH))
```

3. The client's certificate must be mapped to a RACF user ID. Client certificates can be associated with a RACF user ID in two ways. Each expected client certificate can be mapped to a RACF user ID individually, or groups of certificates that match a specified pattern can be mapped to a RACF user ID. This latter approach significantly reduces the administration burden of managing digital certificates and is the approach that is described here.

Use the RACDCERT MAP command to define a certificate name filter. This process maps the certificate subject's distinguished name (DN) to a RACF user ID. Certificate name filtering supports generic filters that allow multiple certificates to be associated with a single RACF user ID.

- a. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

- b. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

```
RACDCERT MAP ID(clientUserid) SDNFILTER('CN=myClient.host.com.O=IBM.C=US')  
WITHLABEL('certificateLabel')
```

**Important:** In this command, the syntax of the SDNFILTER is significant. Use periods to separate the components of the distinguished name and remove any spaces between DN components.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the *z/OS® Security Server RACF Command Language Reference*.

- c. Refresh the DIGTNMAP RACF class for the changes to take effect.

Enter the following RACF command:

```
SETROPTS RACLIST(DIGTNMAP) REFRESH
```

When this certificate is presented to the CICS region as part of a TLS handshake, the CICS region runs the subsequent transaction under the *clientUserid*, which is associated with the certificate in RACF.

4. Define the TCPIP SERVICE.

A TCPIP SERVICE is needed to listen for connections from the client. The connection uses TLS and requires the client to authenticate with a TLS certificate that is mapped to a user ID in RACF.

```
DEFINE TCPIP SERVICE(tcpipService) GROUP(groupA)  
PORT(tcpipPort)  
SSL(CLIENTAUTH) AUTHENTICATE(CERTIFICATE)  
URM(NONE) PROTOCOL(HTTP) TRANSACTION(CWXN)
```

5. Define the PIPELINE.

The PIPELINE provides information about the message handler programs that act on a service request and on the response.

```
DEFINE PIPELINE(webServicePipe) GROUP(groupA)
  CONFIGFILE(soapProviderXmlFile) SHELF(shelfDir)
  WSDIR(wsProviderDir)
```

The CONFIGFILE specifies the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic. An example is:

```
<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>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.2_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

This sample must be copied into a zfs named *soapProviderXmlFile*.

6. Authorize *assertedUserid* to the pipeline alias transaction CPIH and subsequent processing:

The following commands associate the *assertedUserid* with the *taskUserGroup* and grant the group READ access to the PIPEUSER member list that contains CPIA, CPIH, CPIL, and CPIQ:

```
RDEFINE GCICSTRN PIPEUSER ADDMEM(CPIA,CPIH,CPIL,CPIQ)
UACC(NONE)
PERMIT GCICSTRN PIPEUSER ID(taskUserGroup) ACCESS(READ)
CONNECT assertedUserid GROUP(taskUserGroup)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

If the PIPEUSER profile is already defined and associated with a user group, then you need only to add *assertedUserid* to that group.

7. Install the definitions in group *groupA*. Installing the PIPELINE creates URIMAP and WEBSERVICE resources for the web service.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and URIMAP resources:

```
DFHS00107 02/07/2022 13:27:00 APPLID TCPIPService tcpipService has been opened on port tcpipPort
at IP address ANY.
DFHPI0701 I 02/07/2022 13:27:00 APPLID regionUseridA PIPELINE webServicePipe has been
created.
DFHPI0204 I 02/07/2022 13:27:00 APPLID regionUseridA PIPELINE webServicePipe is now ENABLED and
is ready for use.
DFHPI0703 I 02/07/2022 13:27:00 APPLID regionUseridA PIPELINE webServicePipe is about to scan the
WSDIR directory.
DFHPI0901 I 02/07/2022 13:27:00 APPLID regionUseridA New WEBSERVICE webServiceProvider is being
created during a scan against PIPELINE webServicePipe.
DFHPI0910 I 02/07/2022 13:27:00 APPLID regionUseridA WEBSERVICE webServiceProvider within
PIPELINE webServicePipe has been created.
DFHPI0915 I 02/07/2022 13:27:00 APPLID regionUseridA WEBSERVICE webServiceProvider is now
INSERVICE and is ready for use.
DFHPI0903 I 02/07/2022 13:27:00 APPLID regionUseridA New URIMAP uriMapA is being created during a
scan against PIPELINE webServicePipe for
WEBSERVICE webServiceProvider.
DFHPI0904 I 02/07/2022 13:27:00 APPLID regionUseridA URIMAP uriMapA has been created.
DFHPI0704 I 02/07/2022 13:27:00 APPLID regionUseridA PIPELINE webServicePipe Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The TCPIP SERVICE has an OPENSTATUS of OPEN.
- The PIPELINE has an ENABLESTATUS of ENABLED.
- The WEBSERVICE has a STATE of INSERVICE.
- You have a generated URIMAP with an ENABLESTATUS of ENABLED.

To validate the security environment is functioning correctly when you run this example, you can use the CICS security request recording (SRR) utility. The SRR logs all authorization calls and the matching response codes.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **Web** tab and set the **TCPIP Service** field to *tcpipService*. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

The web service client can now start the CICS web service provider.

## Designing security for CICS web service requesters

Typically, a CICS web service requester sends a request to the web service provider. To secure this scenario, consider the implications for different aspects of security and decide which options are the best for you. Examples illustrate some recommended options.

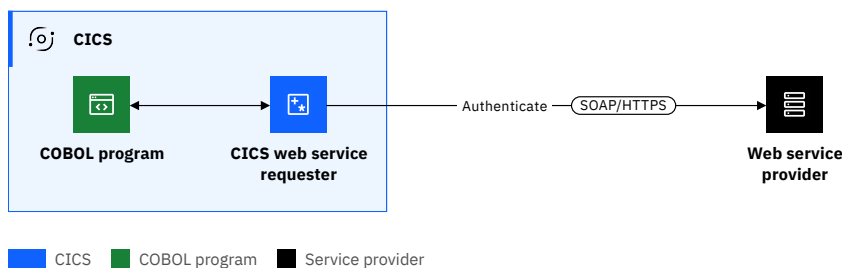
For the equivalent considerations for web service requester applications, see [Designing security for CICS web service providers](#).

- [“Scenario for CICS web service requesters” on page 417](#)
- [“Security design considerations for CICS web service requesters” on page 417](#)

For more information about configuring security for a CICS web service requester, see [Configuring security for CICS web services](#).

### Scenario for CICS web service requesters

In [Figure 38 on page 417](#), the CICS web service requester sends a request to a web service provider.



*Figure 38. Direct request to the web service provider*

### Security design considerations for CICS web service requesters

When you design security for CICS web service requester scenarios, consider the implications for:

- [Authentication and identification](#)
- [Confidentiality and integrity](#)

## Authentication and identification considerations for web service requester security

### Will you use transport-based or SOAP message-based authentication?

You can use transport-based security or SOAP message-based authentication, depending on how the web service provider application expects to receive authentication credentials.

Transport-based authentication can be implemented by using HTTP basic authentication or TLS client authentication.

The XWBAUTH global user exit is used to specify the credentials to send. See [HTTP client send exit XWBAUTH](#).

Web HTTPS is used to connect to a web service provider. The web service provider can require CICS to provide a client certificate. See [Design example: Securing a web service request with TLS client authentication](#).

Use SOAP message-based authentication (WS-Security) when the web service provider expects to receive an authentication token in a SOAP header.

- If you choose SOAP message-based security, you can configure the CICS security handler to include one of the following authentication tokens in the security header of outbound SOAP message:
  - Username tokens that contain only a username.
  - X.509 digital certificate

6.3

Beta

Support for X.509 certificate authentication for the WS-Security feature is removed as of CICS TS 6.3.

If you use other token types, you can configure the CICS security handler to call a [Security Token Service \(STS\)](#). Alternatively you can write your own message handler to use the Trust client interface.

**Recommended:** Use TLS to secure the connection to the STS.

- In some scenarios, SOAP message authentication can be used along with transport-based authentication, for example, the RACF user ID of the CICS task can be flowed in a SOAP security header and the CICS region can be authenticated to the web service provider by using TLS client authentication. See [Design example: Securing a web service request with identity assertion](#).

## Confidentiality and integrity considerations for web service requester security

6.1

6.2

### TLS, XML signatures and XML encryption, or non-encrypted?

TLS can be used for integrity and confidentiality for point-to-point connections where no intermediate server is involved, but it does not ensure end-to-end message integrity or encryption because the message is not protected within intermediate servers.

XML signatures and XML encryption can be used to ensure integrity and confidentiality end-to-end, including within intermediate servers. However, this capability is rarely used because the performance overhead of XML signature processing and XML encryption in CICS is significant. As an alternative, these capabilities are often implemented in an intermediate server such as IBM DataPower Gateway.

**Recommendation:** If you need XML signature processing or XML encryption, consider implementing these capabilities in an intermediate server, such as IBM DataPower Gateway, instead of directly in CICS.

6.3

Beta

Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3. You can still use Transport Layer Security (TLS) to secure SOAP messages. As a result, the XML Toolkit for z/OS is no longer required.

Shown here are some design examples to demonstrate different configurations.

## Design example: Securing a web service request with TLS client authentication

This scenario shows how TLS client authentication is used to secure a web service request from CICS to a web service provider application.

For instructions about how to configure for this scenario, see [Configuration example: Securing a web service request with TLS client authentication](#).

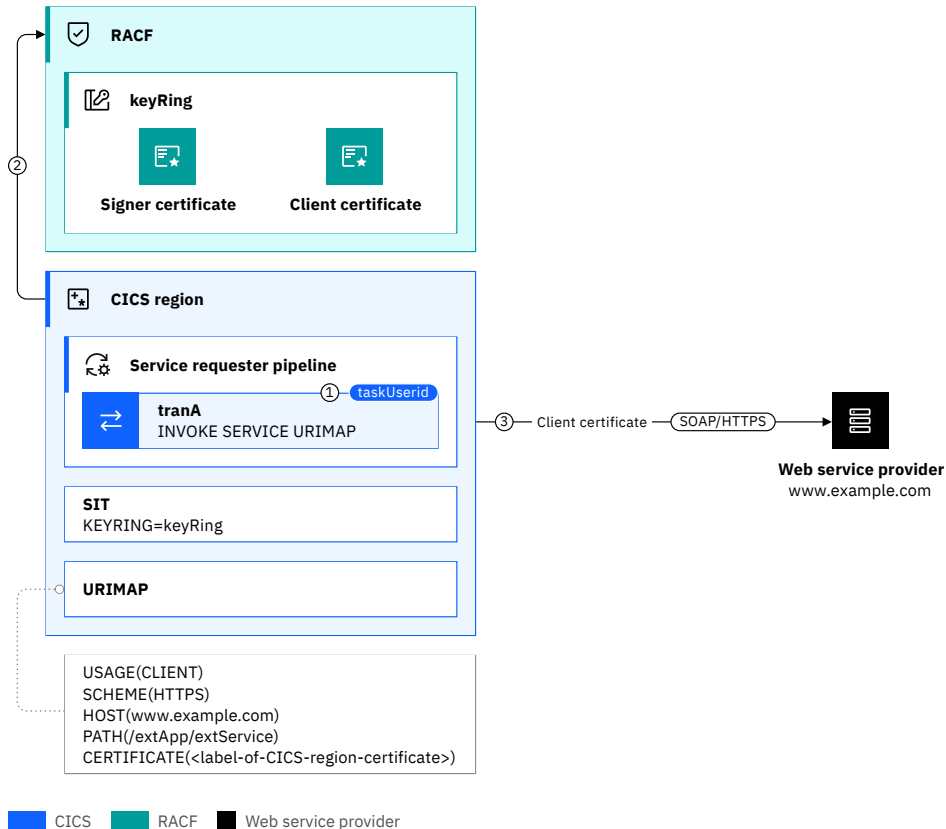


Figure 39. TLS authentication in a web service requester scenario

The URIMAP resource specifies the hostname and path of the external web service. It also specifies SCHEME (HTTPS) so that an HTTPS request is sent, and the label of client certificate that is used in the TLS handshake.

1. The CICS program uses the **EXEC CICS INVOKE SERVICE URIMAP()** command to call the external web service provider.
2. A TLS handshake is performed. The external web service provider sends its server certificate and asks CICS to provide its certificate. CICS validates the chain of trust by checking that the server certificate issuer (signer certificate) is in the RACF key ring (*keyRing*).
3. CICS sends its client certificate that is specified in the URIMAP.

**Recommended:** Use the CIPHERS attribute of the **URIMAP** to define the ciphers that can be used for the SOAP/HTTPS request.

For more information about configuring TLS client authentication for a CICS web service requester, see [CICS as an HTTP client: authentication and identification](#).

## Configuration example: Securing a web service request with TLS client authentication

Learn how to configure a CICS web service requester that authenticates with a web service provider by using a TLS client certificate.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a web service request with TLS client authentication](#).

You must be familiar with web services and TLS.

You must complete these tasks:

- Completed the task [Configuring CICS to use SSL](#).
- Set up a CICS web service requester that is implemented as described in [Developing SOAP web services](#).
- Set up a web service provider that is implemented to be started by the CICS web service requester. For example, a CICS web service provider as described in [Design example: Securing a direct request with TLS client authentication](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to issue web service requests over TLS. The CICS region validates the TLS server certificate that is presented by the started server. The web service requester identifies itself to the provider by using a TLS client certificate.

This task assumes the following definitions:

- *clientCertA* is the TLS certificate to be presented to the web service provider to identify the client
- *datasetName* is the data set that *clientCertA* is exported into to allow it to be imported into the web service provider's key store.
- *webserviceA* is the name of the WEBSERVICE resource that is generated from the WSBIND file by CICS when the pipeline is installed.
- *urimapA* is the name of the URIMAP used to specify the URI of the target web service.
- *channelNameA* is the name of the channel that is used to pass the containers that hold the data that is mapped by the application data structure.
- *operationA* is the name of the operation that is to be started as defined in the WSDL file.
- *groupA* is the RDO group name.
- *pipelineA* is the name of a PIPELINE definition.
- *webserviceURIHost* is the hostname for the web service to be requested.
- *webserviceURIPort* is the port for the web service to be requested.
- *webserviceURIPath* is the path part of the URI for the web service to be requested.
- *soapRequesterXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.
- *matchID* an identifier for the Security Request Recording (SRR) report

- *txnA* the transaction that starts the web service request.
- *regionUseridA* is the region user ID for the CICS region.
- *taskUserID* is the user that triggers the web service request.

## Procedure

1. Export certificate *clientCertA* to *dataset* and transfer it to the platform on which the web service provider is implemented.

```
RACDCERT EXPORT (LABEL('clientCertA')) DSN('datasetName') -
          FORMAT(CERTDER)
```

2. Import the client certificate into the web service provider so it can be used to identify the web service requester. How this import is performed depends on the platform on which the web service provider is implemented.
3. Ensure that the WSBIND file in the *wsProviderDir* directory specifies the correct URI for the web service to be called. As the call is made over a TLS encrypted connection, the URI should start with https (rather than http). This configuration might require the source WSDL file being updated and the tools rerun to generate the updated WSBind file. Options for generating the WSDL file can be found in [Developing SOAP web services](#).
4. Ensure that the code that starts the web service specifies the URIMAP to be used, for example:

```
EXEC CICS INVOKE SERVICE(webserviceA)
          URIMAP(urimapA)
          CHANNEL(channelNameA)
          OPERATION(operationA)
          RESP(resp) RESP2(resp2)
```

5. Update the targeted web service provider to require a TLS connection with the requester that identifies itself with a client certificate.
6. Define the pipeline.

The PIPELINE provides information about the message handler programs that act on an outgoing service request and on the response.

```
DEFINE PIPELINE(pipelineA) GROUP(groupA)
        CONFIGFILE(soapRequesterXmlFile) SHELF(shelfDir)
        WSDIR(wsProviderDir)
```

An example CONFIGFILE can be found in `<cics-install-dir>/samples/pipelines/basicsoap11requester.xml`.

This configuration is not adding any security to this scenario. The security in this scenario is provided by the encryption of the communications by using TLS and the checking of the TLS certificates at either end of the connection.

7. Define the URIMAP.

The URIMAP matches the URI of the web service request and provides information on how to process the requests. 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.

```
DEFINE URIMAP(urimapA) GROUP(groupA)
          USAGE(CLIENT) SCHEME(HTTPS)
          PORT(webserviceURIPort) HOST(webserviceURIHost)
          PATH(webserviceURIPath)
          CERTIFICATE(clientCertA)
          AUTHENTICATE(No)
```

8. Install the definitions in group *groupA*.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and a URIMAP resource:

```
DFHWP1560 03/25/2022 21:21:18 APPLID regionUseridA URIMAP urimapA has been
created.
DFHPI0701 I 03/25/2022 21:21:18 APPLID regionUseridA PIPELINE pipelineA has been
created.
DFHPI0204 I 03/25/2022 21:21:18 APPLID regionUseridA PIPELINE pipelineA is now ENABLED and is
ready for use.
DFHPI0703 I 03/25/2022 21:21:18 APPLID regionUseridA PIPELINE pipelineA is about to scan the
WSDIR directory.
DFHPI0901 I 03/25/2022 21:21:18 APPLID regionUseridA New WEBSERVICE webserviceRequesterA is being
created during a scan against
pipelineA.
```

```
DFHPI0910 I 03/25/2022 21:21:18 APPLID regionUseridA WEBSERVICE webserviceRequesterA within
PIPELINE pipelineA has been created.
DFHPI0915 I 03/25/2022 21:21:18 APPLID regionUseridA WEBSERVICE webserviceRequesterA is now
INSERVICE and is ready for use.
DFHPI0704 I 03/25/2022 21:21:18 APPLID regionUseridA PIPELINE pipelineA Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The PIPELINE should have an ENABLESTATUS of ENABLED.
- The URIMAP should have an ENABLESTATUS of ENABLED.
- The WEBSERVICE should have a STATE of INSERVICE.

To validate the security environment is functioning correctly, you need a web service requester that is configured to authenticate by using a TLS client certificate. This web service needs to start an appropriate web service provider.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** context menu option. On that dialog, select the **All** tab and set the **Transaction ID** *tranA*. See [Checking that a CICS security configuration example is working by using the SRR](#) for details.

The web service requester can now start the CICS web service provider.

## Design example: Securing a web service request with identity assertion

This scenario shows how you can use identity assertion to secure a web service request from CICS to a web service provider application.

For instructions about how to configure for this scenario, see the [Configuration example: Securing a web service request with identity assertion](#).

Figure 40 on page 423 shows a scenario where CICS asserts the user ID of the running CICS task to the web service provider. TLS client authentication is used in this scenario to establish trust between CICS and the external web service provider.

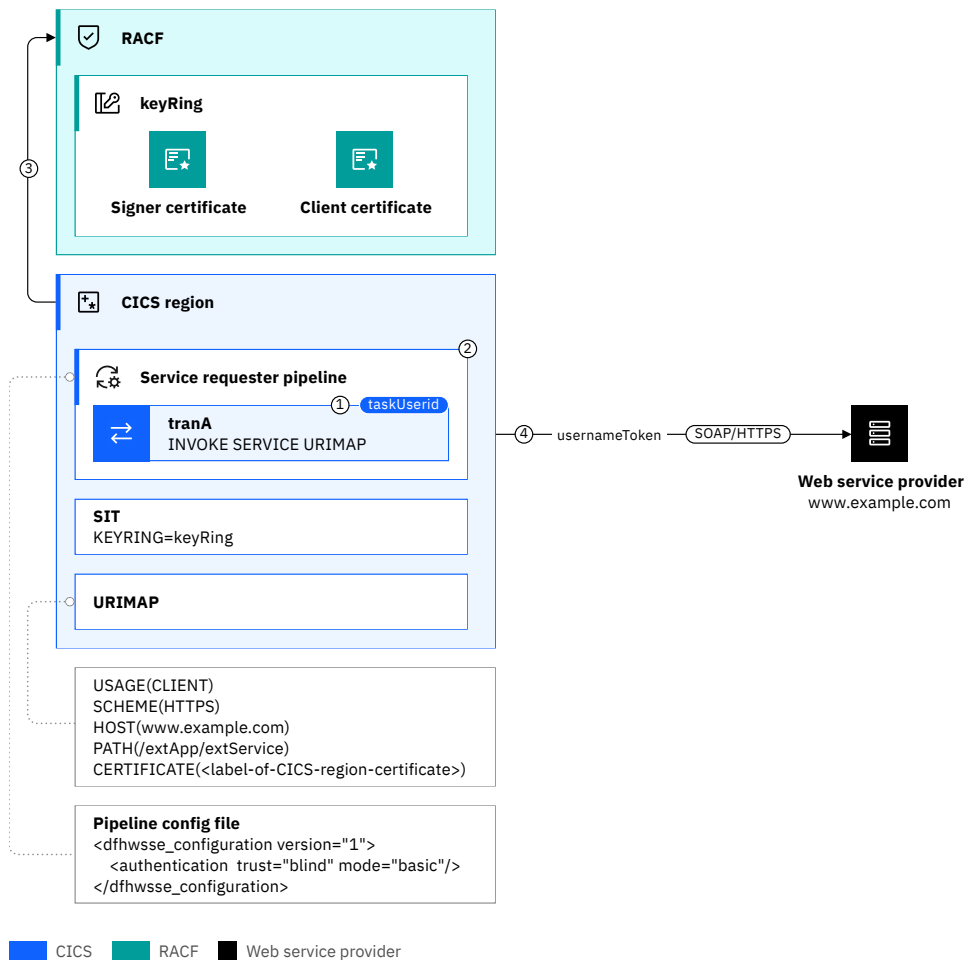


Figure 40. Identity assertion in a web service requester scenario

The web service requester pipeline includes the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic.

The URIMAP resource specifies the hostname and path of the external web service. It also specifies SCHEME (HTTPS) so that an HTTPS request is sent, and the label of client certificate that is used in the TLS handshake.

In Figure 40 on page 423:

1. The CICS program uses the **EXEC CICS INVOKE SERVICE URIMAP()** command to call the external web service provider.
2. The service requester pipeline adds an identity token (*usernameToken*) to the SOAP message. 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.
3. A TLS handshake is performed. The web service provider sends its server certificate and asks CICS to provide its client certificate. CICS validates the chain of trust by checking that the server certificate issuer (signer certificate) is in the RACF key ring (*keyRing*).
4. The SOAP request with a *usernameToken* containing the task user ID is sent over HTTPS to the web service provider.

## Configuration example: Securing a web service request with identity assertion

Learn how to configure a CICS web service requester that asserts an identity on the request to a web service provider.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a web service request with identity assertion](#).

You must be familiar with web services and TLS.

You must complete these tasks:

- Completed the task [Configuring CICS to use SSL](#).
- Set up a CICS web service requester that is implemented as described in [Developing SOAP web services](#).
- Set up a web service provider that is implemented to be started by the CICS web service requester. For example, a CICS web service provider as described in [Configuration example: Asserting an identity to the CICS web service provider](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to issue web service requests over TLS. The CICS region validates the TLS server certificate that is presented by the started server. The web service requester identifies itself to the provider by using a TLS client certificate.

This task assumes the following definitions:

- *clientCertA* is the TLS certificate to be presented to the web service provider to identify the client.
- *datasetName* is the data set that *clientCertA* is exported into to allow it to be imported into the web service provider's key store.
- *webserviceA* is the name of the WEBSERVICE resource that is generated from the WSBIND file by CICS when the pipeline is installed.
- *urimapA* is the name of the URIMAP used to specify the URI of the target web service.
- *channelNameA* is the name of the channel that is used to pass the containers that hold the data that is mapped by the application data structure.
- *operationA* is the name of the operation that is to be started as defined in the WSDL file.
- *groupA* is the RDO group name.
- *pipelineA* is the name of a PIPELINE definition.
- *webserviceURIHost* is the hostname for the web service to be requested.
- *webserviceURIPort* is the port for the web service to be requested.
- *webserviceURIPath* is the path part of the URI for the web service to be requested.
- *soapRequesterXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.
- *matchID* an identifier for the security request recording (SRR) report.

- *regionUseridA* is the user ID that the CICS region is running under.
- *taskUserID* is the user that triggers the web service request.

## Procedure

1. Export certificate *clientCertA* to *dataset* and transfer it to the platform on which the web service provider is implemented.

```
RACDCERT EXPORT (LABEL('clientCertA')) DSN('datasetName') -
          FORMAT(CERTDER)
```

2. Import the client certificate into the web service provider so it can be used to identify the web service requester. How this import is performed depends on the platform on which the web service provider is implemented.
3. Ensure that the WSBIND file in the *wsProviderDir* directory specifies the correct URI for the web service to be called. As the call is made over a TLS encrypted connection, the URI should start with https (rather than http). This configuration might require the source WSDL file being updated and the tools rerun to generate the updated WSBind file. Options for generating the WSDL file can be found in [Developing SOAP web services](#).
4. Ensure that the code that starts the web service specifies the URIMAP to be used, for example:

```
EXEC CICS INVOKE SERVICE(webserviceA)
          URIMAP(urimapA)
          CHANNEL(channelNameA)
          OPERATION(operationA)
          RESP(resp) RESP2(resp2)
```

5. Update the targeted web service provider to require a TLS connection with the requester that identifies itself with a client certificate. The targeted web service provider should also be configured for identity assertion with blind trust and an authentication mode of basic.
6. Define the pipeline.

The PIPELINE provides information about the message handler programs that act on an outgoing service request and on the response.

```
DEFINE PIPELINE(pipelineA) GROUP(groupA)
        CONFIGFILE(soapRequesterXmlFile) SHELF(shelfDir)
        WSDIR(wsProviderDir)
```

The CONFIGFILE specifies the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic. An example is:

```
<requester_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="blind" mode="basic"/>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
  </service>
</requester_pipeline>
```

This sample must be copied into a zfs file named *soapProviderXmlFile*.

This configuration asserts that a user ID is asserted to the web service provider, but no password is passed. The user ID is either explicitly specified in the container DFHWS-USERID, or if that is not specified then it is *taskUserID*.

7. Define the URIMAP.

The URIMAP matches the URI of the web service request and provides information on how to process the requests. 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.

```
DEFINE URIMAP(urimapA) GROUP(groupA)
  USAGE(Client) SCHEME(HTTPS)
  PORT(webserviceURIPort) HOST(webserviceURIHost)
  PATH(webserviceURIPath)
  CERTIFICATE(clientCertA)
  AUTHENTICATE(No)
```

8. Install the definitions in group *groupA*.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and a URIMAP resource:

```
DFHWB1560 04/05/2022 15:26:10 APPLID regionUserida URIMAP urimapA has been
created.
DFHPI0701 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA has been
created.
DFHPI0204 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA is now ENABLED and is
ready for use.
DFHPI0703 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA is about to scan the
WSDIR directory.
DFHPI0901 I 04/05/2022 15:26:10 APPLID regionUserida New WEBSERVICE webserviceRequesterA is being
created during a scan against PIPELINE
pipelineA.
```

```
DFHPI0910 I 04/05/2022 15:26:10 APPLID regionUserida WEBSERVICE webserviceRequesterA within
PIPELINE pipelineA has been created.
DFHPI0915 I 04/05/2022 15:26:10 APPLID regionUserida WEBSERVICE webserviceRequesterA is now
INSERVICE and is ready for use.
DFHPI0704 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The PIPELINE should have an ENABLESTATUS of ENABLED.
- The URIMAP should have an ENABLESTATUS of ENABLED.
- The WEBSERVICE should have a STATE of INSERVICE.

To validate the security environment is functioning correctly, you need a web service requester that is configured to authenticate by using a TLS client certificate. This web service needs to start an appropriate web service provider.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** context menu option. On that dialog, select the **All** tab and set the **Transaction ID** *txnA*. See [Checking that a CICS security configuration example is working by using the SRR for details](#).

The web service requester can now start the CICS web service provider.

## Configuring SOAP message security for CICS web services

Before you configure SOAP message security for CICS web services, make sure that you are familiar with the information in [How it works: SOAP message security](#), [Designing security for CICS web service providers](#), and [Designing security for CICS web service requesters](#).

These are common configuration examples that are based on the Design examples:

- Web service provider examples.
  - [Configuration example: Securing a direct request with TLS client authentication](#)
  - [Configuration example: Asserting an identity to the CICS web service provider](#)

- Web service requester examples.
  - [Identity assertion](#)
  - [Identity assertion](#)

When you have chosen how to secure web services, follow the instructions for the relevant tasks from Table 20 on page 427.

Activity	For web service providers	For web service requesters	Instructions
<b>Review prerequisites for implementing WS-Security</b>	✓	✓	<a href="#">Installing the prerequisites for WS-Security support</a>
<b>Configure a pipeline for SOAP message security</b>	✓	✓	<a href="#">Configuring the pipeline for Web Services Security</a>
<b>Configure ICRX-based identity propagation</b>	✓	n/a	<a href="#">Configuring provider mode web services for identity propagation</a>
<b>Sign or encrypt SOAP messages</b>	✓	✓	<a href="#">Configuring RACF for Web Services Security</a> and the instructions for signing or encrypting SOAP messages in <a href="#">Configuring the pipeline for Web Services Security</a>
<b>Use the Trust client interface to invoke a Security Token Service (STS)</b>	✓	✓	<a href="#">Invoking the Trust client from a message handler</a>
<b>Write a custom security handler to secure SOAP messages</b>	✓	✓	<a href="#">Writing a custom security handler</a>

## Installing the prerequisites for WS-Security support

To implement WS-Security support in CICS, you must install the IBM XML Toolkit for z/OS and add 3 libraries to the DFHRPL concatenation.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices](#) and [discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

### Procedure

1. Install the free IBM XML Toolkit for z/OS.

You can download it from the following site: <https://www.ibm.com/servers/eserver/zseries/software/xml/>.

- **6.2** You must install version 1.11.
- **6.1** You must install version v1.10.

Other versions of the IBM XML Toolkit for z/OS do not work with Web Services Security support in CICS.

6.3 Beta Support for installation is removed as of CICS TS 6.3.

2. 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 CICSTS64 for CICS TS beta.

The first two libraries contain DLLs that are required at run time by the CICS security handler. IXM4C58 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.

3. 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.

## Configuring the pipeline for WS-Security

To configure a pipeline to support WS-Security, 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. Remember that CICS does not support WS-Policy and ignores any security constraints that you configure in the WSDL document.

For information about the elements of pipeline configuration files, see [Support for WS-Security](#).

## Procedure

1. Add a `<wsse_handler>` element to your pipeline for a security handler.

For information about the structure of this element, see [<wsse\\_handler> pipeline configuration element](#). The security handler must be included in the `<service_handler_list>` element in a service provider or requester pipeline. The order of handler elements in the `<service_handler_list>` element determines the order that each handler is called at run time. In a pipeline that 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.

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.

For information about the structure of this element, see [The <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.
- b) Optional: If you specified **trust=none**, code the **mode** attribute to specify how credentials found in the message are processed.
- c) In the <authentication> element, code these elements:

- i) 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.

- ii) 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`.
- iii) 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.

Do not code both in your pipeline configuration file. For information about the structure of this element, see [The <sts\\_authentication> element](#). 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.
- b) Within the <sts\_authentication> element, code these elements:
  - i) 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 [The <auth\\_token\\_type> element](#).
    - 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
  - ii) 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 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 an IBM 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.

6.3 Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

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>
```

## 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 TLS connection from IBM DataPower Gateway. 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

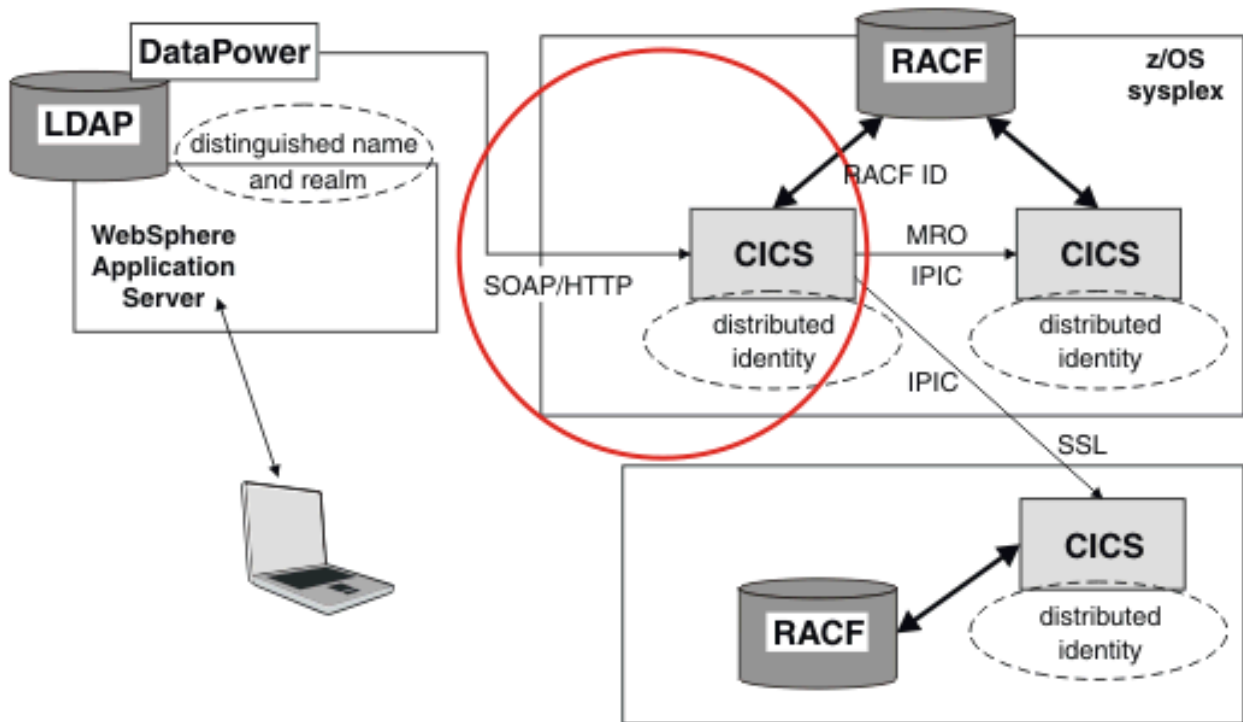
Configure your RACF RACMAP settings before you configure your web service connections. Otherwise, you receive the RACF ICH408I message for every unmapped request that is sent to RACF. For more information about configuring the RACF **RACMAP** command, see [Configuring RACF for identity propagation](#).

You must configure a *trust* relationship between the IBM DataPower Gateway appliance and CICS, for example, using TLS client certification between IBM DataPower Gateway and CICS. The digital certificate that IBM DataPower Gateway 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 security](#).

### About this task

This task explains how to use CICS with a IBM DataPower Gateway 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 41. Configuring CICS to expect an ICRX identity token from IBM DataPower Gateway.*



IBM DataPower Gateway acts as an intermediary between CICS and other applications. Remote web service requester applications connect to the IBM DataPower Gateway appliance using the SOAP protocol. IBM DataPower Gateway 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 TLS 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 IBM DataPower Gateway. The PIPELINE configuration file specifies *blind* trust, because the only possible client is the IBM DataPower Gateway appliance, and IBM DataPower Gateway is communicating with CICS over a secure TLS 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](#).

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 IBM DataPower Gateway is configured to be able to send ICRX information. See [Identity propagation](#).

## Results

Web service requests from IBM DataPower Gateway with an ICRX identity token in the WS-Security header, connected over a client-certified TLS connection, can now flow.

## Configuring RACF for WS-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.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

## 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](#).

## 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 security handler. You can use the Trust client instead of the CICS security handler or in addition to it.

### 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-STSUR, 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](#).

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 security handler. In service provider pipelines, define your message handler at the beginning of the pipeline but after the CICS security handler.

## Writing a custom security handler

To use your own security procedures and processing, write a custom message handler to process secure SOAP messages in the pipeline.

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. The message handler must also be able to cope with security on inbound and outbound messages.

Here is a likely set of steps that your security handler would implement:

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:
  - 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.
  - 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.
  - You might also want to write an audit record each time a service is requested, for example, you could write a message to a CICS user journal.

- To perform advanced authentication, either by exchanging or validating a range of tokens with a Security Token Service, use the Trust client interface which enables you to interact with the STS directly. . See [“Invoking the Trust client from a message handler”](#) on page 435 for details.
  - Validate the credentials of the digital certificate if the message is signed.
  - If parts of the message are encrypted, decrypt the message using the information in the security header. The [How CICS complies with Web Services Security specifications](#) specification provides information about how to do this
4. 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.

For examples of custom message handlers, see [IBM Redbooks: Implementing CICS Web services](#).



---

## Chapter 7. Troubleshooting web services

The problems that you might get when implementing web services in CICS can occur during the deployment process, or at run time when CICS is transforming the messages.

### Related information

[CICS and web services](#)

[Developing JSON web services](#)

[Developing SOAP web services](#)

---

## 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.

- a) 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.

- b) 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.

- c) 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 300 MB, although some documents might require 400 MB.

- a) Increase the region size, or consider setting the region size to 0M.
- b) Check for any active IEFUSI exits, which can limit the region size.

**Note:** If you use a 64bit JVM, ensure you specify a suitable MEMLIMIT value.

- 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.
  - a) Increase the time by coding the **TIME** parameter on the EXEC statement of your job to **TIME=1440**, or increase the MAXCPUIME 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.
  - a) Check that you have authorized CICS to read the web service binding file in z/OS UNIX.
  - b) 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.
  - c) Check that two web service binding files with the same name are not in different pick up directories.
  - d) 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.
  - e) 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.
  - f) 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.
  - g) 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.
  - a) If you received a DFHPI0700 error message, you need to enable PL/I language support in your CICS region.

This is required before you can install any PIPELINE resources. See [Language Environment support for PL/I](#) for more information.
  - b) Check that you have authorized CICS to access the z/OS UNIX directories to read the pipeline configuration files.
  - c) 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.

- d) 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.
  - a) 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.
  - b) Check that you do not have any unprintable characters, such as tabs, in the pipeline configuration file.
  - c) 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.
  - d) 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.

  - a) 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.
  - b) 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 IBM MQ transport error has occurred.
 

If the transport is HTTP, the client receives a 500 Server Internal Error message. If the transport is IBM 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 DFHxx message and a 404 Not Found error message.
  - a) 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.
  - b) 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 439](#).
  - c) 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.

- a) 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.

- b) 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 [CEDX](#).

- c) 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 445](#).
- If you think your problem is related to MTOM messages, see [“Diagnosing MTOM/XOP errors” on page 444](#).
- If a persistent HTTP connection is periodically closed:
  - a) Check whether performance tuning for HTTP connections is enabled. For more information, see [SOTUNING](#).
  - b) 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.
  - a) Check whether performance tuning for HTTP connections is enabled. For more information, see [SOTUNING](#).
  - b) If performance tuning is enabled, when CICS is at maximum capacity all inbound HTTP connection open requests will queue outside of CICS in the TCPIP SERVICE'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.
  - c) Use **NETSTAT ALL** to obtain the ConnectionsDropped statistics for the TCPIP SERVICE's listening connection. This is the number of connection requests received and dropped because the number of connection requests waiting in the backlog queue has reached the maximum limit. For more information, see [Netstat in z/OS Communications Server: IP System Administrator's Commands](#).

Statistics about connections dropped are also available from the following fields in the [TCP/IP services: Resource statistics](#):

Connections dropped (SOR\_CONNS\_DROPPED)

Time connection last dropped (SOR\_CONN\_LAST\_DROPPED)

- d) If the ConnectionsDropped value is too high, consider increasing the TCPIP SERVICE'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.
  - a) 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.
  - b) 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.
  - a) Look at the contents of the DFHERROR container.

This should contain an error message and some data describing the problem that occurred.
  - b) 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](#) for details.
  - c) 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.
  - d) 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 445.
  - e) 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.
  - a) Check that the network end point is present.
  - b) 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.
  - a) 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”](#) on page 444.

## Diagnosing MTOM/XOP errors

MTOM/XOP errors can occur at run time, in both requester and provider mode pipelines.

If you are having problems configuring a pipeline to support MTOM/XOP, follow [“Diagnosing deployment errors”](#) on page 439.

### 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.
  - a) 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.
  - b) 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.
  - c) 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.
  - d) 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.
  - e) 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.
  - a) Look at the contents of the DFHERROR container.

This should contain an error message and some data describing the problem that occurred.
  - b) 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:

- a) If you have auxiliary trace active in your CICS region, check for any exception trace entries.
- b) 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 *
+0000112 in* 2-0000 D4C9D4C5 40A2A895 A381A740 85999996 994081A3 404EF0F0 F0F0F1F1 F2408995 *MIME syntax error at
header *
3-0000 40C39695 A38595A3 60A3A897 85408885 81848599 * Content-type
type="application/xop* 0020 5F626F75 6E646172 793B2074 7970653D 22617070 6C696361 74696F6E 2F786F70 *_boundary;
info="application/s* 0020 2B786D6C 223B2073 74617274 2D696E66 6F3D2261 70706C69 63617469 6F6E2F73 **xml"; start-
start= 0040 6F61702B 786D6C22 3B207374 6172743D *oap+xml";
* <userabc@myzos.example.com>* 4-0000 3C736F61 70736C75 6E674074 6573742E 68757273 6C65792E 69626D2E 636F6D3E
charset=utf-8 0020 3B206368 61727365 743D7574 662D38 *;
```

- 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.

- a) 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.
- b) To find the exact cause of the problem, check for any exception trace points.

In particular, look for trace points beginning with PI 13xx. 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.

## Diagnosing data conversion errors

Data conversion errors can occur at run time when converting a SOAP message into a CICS COMMAREA or container and from a COMMAREA or container into a SOAP message. Symptoms include the generation of SOAP fault messages and CICS messages indicating that a failure has occurred.

### Procedure

If you have a data conversion problem, perform the following steps:

1. Ensure that the WEBSERVICE resource is up to date. Regenerate the web service binding file for the web service and redeploy it to CICS.

2. Ensure that the remote web service has been generated using the same version of the web service document (WSDL) as used or generated by CICS.
3. If you are sure that the WEBSERVICE resource is using a current web service binding file:
  - a) Enable runtime validation for the WEBSERVICE resource using the command `SET WEBSERVICE(name) VALIDATION` where *name* is the WEBSERVICE resource name.
  - b) Check for the CICS messages DFHPI1001 or DFHPI1002 in the message log.  
DFHPI1001 describes the precise nature of the data conversion problem and 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

Data conversion errors can occur at run time during the conversion between a SOAP message and a CICS COMMAREA or container. When data conversion errors occur, SOAP fault messages are issued.

### When CICS is a web service provider

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.

### When CICS is a web service requester

If CICS is the web service requester, the **INVOKE SERVICE** command returns a RESP code of INVREQ and a RESP2 value of 14.

#### Related concepts

[“Why data conversion errors occur” on page 446](#)

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.

## Troubleshooting JSON web services

---

The problems that you might get when implementing JSON web services in CICS can occur during the deployment process, or at run time when CICS is transforming the messages. This problem determination information is specific to CICS.

Support for CICS as a service provider for JSON requests is heavily based on CICS support for SOAP web services.

## Troubleshooting JSON deployment problems

Deployment errors can occur when you try to either install a PIPELINE or a WEBSERVICE resource in CICS. The most common deployment errors are described here, including the symptom of the problem, the cause, and the solution.

### Procedure

The following errors might occur when you install a PIPELINE or WEBSERVICE resource in CICS:

- You receive a DFHPI0914 error message when you attempt to install a WEBSERVICE resource. The message includes some information about the cause of the installation failure.
  - a) Check that you have authorized CICS to read the web service binding file in z/OS UNIX.
  - b) Check that the web service binding file is not corrupted.

This corruption can occur, for example, if you use FTP to transfer the file to z/OS UNIX in text mode rather than binary mode.
  - c) Check that two web service binding files with the same name are not in different pickup directories.
  - d) 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.
  - e) If you are using DFHJS2LS or DFHLS2JS, check the messages that are issued by the job to see whether there are any problems that you must resolve before creating the WEBSERVICE resource.
- The PIPELINE resource fails to install and you receive a DFHPI0700, DFHPI0712, DFHPI0714 or another error message that states that the PIPELINE resource failed.
  - a) If you received a DFHPI0700 error message, you must enable PL/I language support in your CICS region.

This support is required before you can install any PIPELINE resources. See [Language Environment support for PL/I](#) for more information.
  - b) Check that you have authorized CICS to access the z/OS UNIX directories to read the pipeline configuration files.
  - c) 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.
  - d) 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.
  - a) 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 that you have coded it in the correct place.
  - b) Check that you do not have any unprintable characters, such as horizontal tab characters, in the pipeline configuration file.
  - c) 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.
  - d) 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.

## Troubleshooting the JSON assistant

If you are experiencing problems with the JSON assistant, use troubleshooting techniques to diagnose the problem.

### Procedure

The following errors might occur when you run the JSON assistant:

- You receive a return code of 0, 4, 8, or 12 when running the CICS JSON assistant batch jobs.  
For more information about the return codes, see [“JSON assistant return codes”](#) on page 451.
  - a) Check the job log for any warning or error messages.  
Look up the detailed explanations for the messages. The explanations often describe actions that you can take to fix the problem.
  - b) 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 are case sensitive.
  - c) Ensure that you have specified the correct combination of parameters.
- You receive a return code of 1, 136, or 139 when running the CICS JSON assistant batch jobs.  
These return codes indicate that the JVM has failed, usually because there is insufficient storage available. The CICS assistant requires a JCL region size of at least 300 MB.
  - a) Increase the region size, or consider setting the region size to 0M.
  - b) Check for any active IEFUSI exits, which can limit the region size.
- You receive a return code of 137 when running the CICS JSON assistant batch job. This return code means that the job timed out.
  - a) Increase the time by coding the **TIME** parameter on the EXEC statement of your job to **TIME=1440**, or increase the **MAXCPU** value in the SYS1.PARMLIB(BPXPRMxx) member.
- You receive a message in the range DFHPI9700 - DFHPI9711 indicating an invalid or unsupported JSON schema when running DFHJS2LS.
  - a) Check that your JSON schema is valid. You could do this by checking it against the [JSON Schema Validation](#), or by using a JSON validation tool.
  - b) Check that your JSON schema is supported by DFHJS2LS. For more information, see [High-level language and JSON schema mapping](#).

### What to do next

After you have resolved the problem, rerun your JSON assistant batch job.

## Troubleshooting problems with JSON requests

If CICS is rejecting JSON, whether as an input message to a web service or when used with the linkable interface to transform JSON, the JSON might be invalid or not conforming to the schema.

### About this task

If CICS encounters a problem with JSON provided by a web service requester or an application program, an error message is return with details of the problem. When CICS is acting as a web service provider, the error message is returned to the client application as a JSON response. For more information, see [“Error responses returned to the client”](#) on page 450. When an application calls the linkable interface to transform JSON, an error code is returned in the DFHJSON-ERROR container and a detailed message in the DFHJSON-ERRORMSG container. For more information, see [JSON transformer linkable interface](#). In both cases the troubleshooting steps are the same and depend on the type of error.

## Procedure

- Check that your JSON conforms to the restrictions imposed by CICS. For more information, see [JSON web service restrictions](#).

- If a JSON parsing error occurs, check that the JSON is well-formed. The error message provides details of the problem, such as:

```
"Expected a ', ' or '}' at character 44 of {"inquireCatalogRequest":  
"myData }
```

You could use a tool to validate the JSON syntax, for example [JSONLint](#).

- If CICS detects a structural error when it is trying to map JSON to application data, a DFHPI1007 message is issued providing details of the error.

For example:

```
DFHPI1007 04/19/2013 15:14:42 IYK2ZKE1 00112 JSON to data transformation  
failed because of incorrect input (UNDEFINED_ELEMENT Operation) for  
WEBSERVICE SimpleMappings.
```

Check that the JSON contains the JSON objects and properties that are described by the schema you provided to DFHJS2LS, or that was generated by DFHLS2JS. You could use a tool to validate that the JSON conforms to the schema.

- If no error occurs, but your application receives empty data for some fields, check that the corresponding JSON was provided. CICS will not detect the absence of JSON properties described by the schema. You could use a JSON validation tool to check this as described earlier.
- An Internal server error is encountered.

```
Internal Server Error  
CICS TS: Unhandled Pipeline Error  
Release: 670
```

In the browser, check for message DFHSJ1006. Check that the JVMServer exists and is not disabled.

- In some circumstances, CICS adds a wrapper element to the JSON request or response during internal processing. This will never be visible to the application, but can sometimes occur in error messages.

For example:

```
"Error obtaining parser from data source:Expected a ':'  
after a key at character 25 of {"DFHWrapper": { 12jb01c\":"...
```

In these situations the DFHWrapper element should be ignored when determining the cause of the error.

## What to do next

After you have corrected your JSON, rerun the application.

## Error responses returned to the client

If errors occur during processing in the CICS JSON handler, CICS returns a response to the client containing information about the error. An HTTP status code of 500 (Internal Server Error) is returned, and the message body contains details of the error in JSON format, depending on the type of error that occurred.

If an error occurs before or after CICS has called the Axis2 pipeline, the message contains information similar to the following:

```
{  
  "exception" : {  
    "message" : "An exception has occurred while validating HTTP headers".  
    "class" : "com.ibm.cicsts.axis2.Controller"  }}
```

```
}  
}
```

If an error occurs at any other point during processing, the message contains similar information to a SOAP fault, with a detail section which varies depending on the nature of the error. This might contain a CICS message, such as:

```
{  
  "Fault": {  
    "faultstring": "Conversion from SOAP failed",  
    "detail": {  
      "CICSFault": "DFHPI1007 02/14/2013 17:51:47 IYK2ZKE1 00185 XML to data  
transformation failed because of incorrect input UNDEFINED_ELEMENT startItemRuff)  
for WEBSERVICE json_inquireCatalogWrapper."  
    }  
  }  
}
```

## JSON assistant return codes

If a failure occurs while running the JSON assistant batch jobs, a return code is provided indicating the type of failure. This information is contained in the job log.

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 JSON 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 JSON assistant, other codes are typically issued by BPXBATCH, the JVM, or IEBCGENER.

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 [z/OS UNIX System Services Command Reference](#).

## JSON assistant program return codes

You receive a return code of 0, 4, 8, or 12 when running the JSON assistant batch jobs.

Return code	Description
0	The job completed successfully.
4	Warning. The job completed successfully, but one or more warning messages were 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.

## JSON assistant batch job return codes

You receive a return code of 1, 136, 137, or 139 when running the CICS JSON assistant batch jobs.

Return code	Description
1	The JVM failed, usually because there is insufficient storage available. The CICS assistant requires a JCL region size of at least 300 MB.
136	The JVM failed, usually because there is insufficient storage available. The CICS assistant requires a JCL region size of at least 300 MB.
137	The job timed out.

Return code	Description
139	The JVM has failed, usually because there is insufficient storage available. The CICS assistant requires a JCL region size of at least 300 MB.

---

## Appendix A. XML elements for web services pipeline configuration

The configuration of a pipeline used to handle a web service request is specified in an XML document, known as a *pipeline configuration file*. This reference describes the XML elements that are used for pipeline configuration.

The XML schemas for the pipeline configuration files are in the `/usr/lpp/cicsts/cicstsnn/schemas/pipeline/` directory, where `/usr/lpp/cicsts/cicstsnn` is the default install directory for CICS files on z/OS UNIX and `cicstsnn` reflects the release of CICS TS. For example, the directory is `/usr/lpp/cicsts/cicsts64/schemas/pipeline/` for CICS TS beta.

For an introduction to pipeline configuration files, see [Pipeline configuration files](#).

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. 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.

Pipeline	Schema	Root element
Service provider	Provider.xsd	<provider_pipeline>
Service requester	Requester.xsd	<requester_pipeline>

### Elements used only in service providers

- [<apphandler\\_class>](#)
- [<named\\_transport\\_entry>](#)
- [<provider\\_pipeline>](#)
- [<provider\\_pipeline\\_json>](#)
- [<terminal\\_handler>](#)
- [<transport\\_handler\\_list>](#)

### Elements used only in service requesters

- [<requester\\_pipeline>](#)

### Common elements

- [<addressing>](#)
- [<cics\\_json\\_handler\\_java>](#)
- [<cics\\_soap\\_1.1\\_handler>](#)
- [<cics\\_soap\\_1.1\\_handler\\_java>](#)
- [<cics\\_soap\\_1.2\\_handler>](#)
- [<cics\\_soap\\_1.2\\_handler\\_java>](#)
- [<default\\_http\\_transport\\_handler\\_list>](#)
- [<default\\_mq\\_transport\\_handler\\_list>](#)
- [<default\\_transport\\_handler\\_list>](#)
- [<handler>](#)
- [<jvmserver>](#)
- [<repository>](#)

- [<service>](#)
- [<service\\_handler\\_list>](#)
- [<service\\_parameter\\_list>](#)
- [<transport>](#)

#### Elements for MTOM/XOP configuration

- [<mtom>](#)
- [<cics\\_mtom\\_handler>](#)
- [<dfhmtom\\_configuration>](#)
- [<mime\\_options>](#)
- [<mtom\\_options>](#)
- [<xop\\_options>](#)

#### Elements for WS-Security configuration

##### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

- [<wsse\\_handler>](#)
- [<dfhwsse\\_configuration>](#)
- [<authentication>](#)
- [<auth\\_token\\_type>](#)
- [<encrypt\\_body>](#)
- [<sign\\_body>](#)
- [<sts\\_authentication>](#)
- [<sts\\_endpoint>](#)

## The [<addressing>](#) pipeline configuration 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.

## Example

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 <apphandler\_class> pipeline configuration element

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS web services, z/OS Connect Enterprise Edition, or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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. 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](#).

## Used in:

- Service provider

## Contained by:

- [<provider\\_pipeline>](#) element

## Example

```
<apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
```

## The <authentication> pipeline configuration element

Specifies the use of security tokens in the headers of inbound and outbound SOAP messages.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

6.3 Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

6.1 6.2 Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

### Used in:

- Service provider
- Service requester

### Contained by:

- `<dfhwsse_configuration>` element

### Attributes

#### trust

#### mode

Taken together, the trust and mode attributes specify whether asserted identity is used, and, if 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 21 on page 456](#) and [Table 22 on page 457](#).

trust	mode	Meaning
none	none	No credentials are added to the message
none	basic	<i>Invalid combination of attribute values</i>
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 <code>&lt;certificate_label&gt;</code> element, and the algorithm is specified in the <code>&lt;algorithm&gt;</code> element.
blind	none	<i>Invalid combination of attribute values</i>

Table 21. The **mode** and **trust** attributes in a service requester pipeline (continued)

<b>trust</b>	<b>mode</b>	<b>Meaning</b>
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	<i>Invalid combination of attribute values</i>
basic	none	<i>Invalid combination of attribute values</i>
basic	basic	<i>Invalid combination of attribute values</i>
basic	signature	<i>Invalid combination of attribute values</i>
signature	none	<i>Invalid combination of attribute values</i>
signature	basic	Asserted identity is used. CICS adds the following tokens to the message: <ul style="list-style-type: none"> <li>• The trust token is an X.509 security token.</li> <li>• The identity token is a user name with no password.</li> </ul> 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	<i>Invalid combination of attribute values</i>

Table 22. The **mode** and **trust** attributes in a service provider pipeline

<b>trust</b>	<b>mode</b>	<b>Meaning</b>
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	<i>Invalid combination of attribute values</i>
none	basic-kerberos	<i>Invalid combination of attribute values</i>
none	signature	Inbound messages must contain an X.509 security token that has been used to sign the message body.
blind	none	<i>Invalid combination of attribute values</i>
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 22. The **mode** and **trust** attributes in a service provider pipeline (continued)

<b>trust</b>	<b>mode</b>	<b>Meaning</b>
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	<i>Invalid combination of attribute values</i>
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	<i>Invalid combination of attribute values</i>
basic	basic	Inbound messages must use asserted identity: <ul style="list-style-type: none"> <li>• The trust token is a user name token with a password</li> <li>• The identity token is a second user name token without a password. CICS puts this user name in container DFHWS-USERID.</li> </ul>
basic	basic-ICRX	Inbound messages must use asserted identity: <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>
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: <ul style="list-style-type: none"> <li>• <a href="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#Kerberosv5_AP_REQ1510</a></li> <li>• <a href="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#GSS_Kerberosv5_AP_REQ1510</a></li> <li>• <a href="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#Kerberosv5_AP_REQ4120</a></li> <li>• <a href="http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120">http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120</a></li> </ul> 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 22. The **mode** and **trust** attributes in a service provider pipeline (continued)

<b>trust</b>	<b>mode</b>	<b>Meaning</b>
basic	signature	Inbound messages must use asserted identity: <ul style="list-style-type: none"> <li>• The trust token is a user name token with a password</li> <li>• The identity token is an X.509 certificate. CICS puts the user ID associated with the certificate in container DFHWS-USERID.</li> </ul>
signature	none	<i>Invalid combination of attribute values</i>
signature	basic	Inbound messages must use asserted identity: <ul style="list-style-type: none"> <li>• The trust token is an X.509 certificate</li> <li>• The identity token is a user name token without a password. CICS puts the user name in container DFHWS-USERID.</li> </ul> <p>The identity token and the body must be signed with the X.509 certificate.</p>
signature	basic-ICRX	Inbound messages must use asserted identity. <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>
signature	basic-kerberos	<i>Invalid combination of attribute values</i>
signature	signature	Inbound messages must use asserted identity: <ul style="list-style-type: none"> <li>• The trust token is an X.509 certificate</li> <li>• The identity token is a second X.509 certificate. CICS puts the user ID associated with this certificate in container DFHWS-USERID.</li> </ul> <p>The identity token and the body must be signed with the first X.509 certificate (the trust token).</p>

**Notes:**

1. The combinations of the trust and mode attribute values are checked when the PIPELINE 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 in [VERIFY PHRASE](#).

**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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

### 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 `<auth_token_type>` pipeline configuration element

Specifies what type of identity token is required.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

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>](#) element

### 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.

The configuration elements must be used in the sequential order that they are listed.

6.3

Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

### Example

```
<auth_token_type>
  <namespace>http://example.org.tokens</namespace>
  <element>UsernameToken</element>
</auth_token_type>
```

## The [<cics\\_json\\_handler\\_java>](#) pipeline configuration element

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS web services, [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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 element
- The [<terminal\\_handler>](#) element element

### Contains:

1. A [<jvmserver>](#) element
2. An optional [<repository>](#) element

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

## Example

The following example shows the XML for the Java-based JSON handler and its nested elements. The name of the repository in the example reflects the release of CICS TS: for example, /usr/lpp/cicsts/cicsts64/lib/pipeline/repository is from CICS TS beta.

```
<?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/cicsts64/lib/pipeline/repository</repository>
      </cics_json_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAXIS2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

## The <cics\_mtom\_handler> pipeline configuration 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> element
- <requester\_pipeline> element

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 `<cics_soap_1.1_handler>` pipeline configuration 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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

## 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>` pipeline configuration element

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS [web services](#), [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

### Example

The following example shows the XML for the Java-based SOAP handler and its nested elements:

```
<cics_soap_1.1_handler_java>
  <jvmserver>JVMSEV1</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>` pipeline configuration 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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

### 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>` pipeline configuration element

**Deprecated feature:** Support for the JVMSERVER-based configuration option for the web services data transformation service is deprecated. The deprecated options can be replaced with the use of a non-Java pipeline for CICS web services, [z/OS Connect Enterprise Edition](#), or by writing web applications in [Java](#) or [Node.js](#). For more details, see [Stabilization notices](#).

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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

**Example**

The following example shows the XML for the Java-based SOAP handler and its nested elements:

```
<cics_soap_1.2_handler_java>
  <jvmserver>JVMSEV1</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> pipeline configuration element

---

Specifies the message handlers that are invoked by default when the HTTP transport is in use.

**Used in:**

- Service provider

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.

- Service requester

**Contained by:**

- [<transport>](#) element

**Attributes****follow\_redirects****no**

Automatic redirects of HTTP responses are not allowed.

**yes**

Automatic redirects of HTTP responses are allowed to the information contained in **DFHWS-LOCATION**.

**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> pipeline configuration element

---

Specifies the message handlers that are invoked by default when the IBM MQ transport is in use.

### Used in:

- Service provider

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.

- 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> pipeline configuration element

---

Specifies the message handlers that are invoked by default when any transport is in use.

### Used in:

- Service provider

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>
```

- Service requester

### Contained by:

- <transport> element

## Contains:

- Zero 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 [<dfhwsse\\_configuration>](#) pipeline configuration element

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

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>](#) element

### Attributes

#### 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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

## 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 <dfhmtom\_configuration> pipeline 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

#### 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

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

### 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 <encrypt\_body> pipeline configuration element

---

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

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>](#) element

### 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 algorithms shown in [How it works: SOAP message encryption](#).
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.

The configuration elements must be used in the sequential order that they are listed.

### Example

```
<encrypt_body>
  <algorithm>http://www.w3.org/2001/04/xmlenc#aes256-cbc</algorithm>
  <certificate_label>ENCCERT02</certificate_label>
</encrypt_body>
```

## The [<handler>](#) pipeline configuration 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>](#) element
- [<transport\\_handler\\_list>](#) element
- [<service\\_handler\\_list>](#) element
- [<terminal\\_handler>](#) element
- [<default\\_http\\_transport\\_handler\\_list>](#) element
- [<default\\_mq\\_transport\\_handler\\_list>](#) element

### 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.

The configuration elements must be used in the sequential order that they are listed.

### Example

```
<?xml version="1.0"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/http/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> pipeline configuration 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](#) element
- The [<cics\\_soap\\_1.1\\_handler\\_java> element](#) element
- The [<cics\\_soap\\_1.2\\_handler\\_java> element](#) element
- The [<provider\\_pipeline\\_json> element](#) element

### Example

```
<jvmserver>JVMSERVER_NAME</jvmserver>
```

## The <mime\_options> pipeline configuration 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> element](#)

### Attributes

#### 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`.

## The `<mtom>` pipeline configuration 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>](#) element
- [<cics\\_soap\\_1.2\\_handler\\_java>](#) element

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>JVMSEV1</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 `<mtom_options>` pipeline configuration 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>](#) element

## Attributes

### 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 `<named_transport_entry>` pipeline configuration 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 IBM MQ transport, the named resource is the local input queue on which the request is received.
- For the HTTP transport, the resource is the TCPIP SERVICE that defines the port on which the request was received.

### Used in:

- Service provider

### Contained by:

- `<transport>` element

## Attributes

### type

The transport mechanism with which the named resource is associated:

#### wmq

The named resource is a queue

#### http

The named resource is a TCPIP SERVICE

## Contains:

1. A <name> element, containing the name of the resource
2. An optional <transport\_handler\_list> element. Each <transport\_handler\_list> contains one or more <handler> elements.

If you do not code a <transport\_handler\_list> element, then the only message handlers that are invoked when the named transport is used are those that are specified in the <service> element.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

## Example

```
<named_transport_entry type="http">
  <name>PORT80</name>
  <transport_handler_list>
    <handler><program>HANDLER1</program><handler_parameter_list/></handler>
    <handler><program>HANDLER2</program><handler_parameter_list/></handler>
  </transport_handler_list>
</named_transport_entry>
```

In this example, the message handlers specified (HANDLER1 and HANDLER2) are invoked for messages received on the TCPIP SERVICE with the name PORT80.

## The <provider\_pipeline> pipeline configuration 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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

## Example

```
<provider_pipeline>
  <service>
    ...
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

## The <provider\_pipeline\_json> pipeline configuration 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(JVMSEVER) 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"`.

#### **java\_generator={yes|no}**

Select the type of JSON generator that is used to generate outbound messages.

Possible values are:

#### **yes**

Perform the JSON generation by using Java within the JVM server.

#### **no**

Perform non-Java generation of the JSON message. This is the default.

### Contains:

- A <jvmserver> element, containing the name of the JVMSEVER 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 <repository> pipeline configuration 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 beforehand in the handler XML. If the element is not supplied then, the sample repository will be used. When you install CICS TS, the sample Axis2 repository is installed in /usr/lpp/cicsts/cicstsnn/lib/pipeline/repository, where /usr/lpp/cicsts/cicstsnn/lib/pipeline/repository is the default installation directory for CICS files on z/OS UNIX and *nn* reflects the release of CICS TS. For example, /usr/lpp/cicsts/cicsts64/lib/pipeline/repository is for CICS TS beta.

### Used in:

- Service provider
- Service requester

### Contained by:

- The <cics\_json\_handler\_java> element element
- The <cics\_soap\_1.1\_handler\_java> element element
- The <cics\_soap\_1.2\_handler\_java> element element

### Example

```
<cics_soap_1.1_handler_java>
  <jvmserver>JVMSERV1</jvmserver>
  <repository>/lib/pipeline/repository</repository>
</cics_soap_1.1_handler_java>
```

## The <requester\_pipeline> configuration 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

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

### Example

```
<requester_pipeline>
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
    </service_handler_list>
  </service>
</requester_pipeline>
```

## The <service> pipeline configuration element

---

Specifies the message handlers that are invoked for every request.

### Used in:

- Service provider
- Service requester

### Contained by:

- [<provider\\_pipeline>](#) element
- [<requester\\_pipeline>](#) element

### Contains:

1. [<service\\_handler\\_list>](#) element
2. In a service provider only, a [<terminal\\_handler>](#) element

The configuration elements must be used in the sequential order that they are listed.

### Example

```
<service>
  <service_handler_list>
    ...
  </service_handler_list>
  <terminal_handler>
    ...
  </terminal_handler>
</service>
```

## The <service\_handler\_list> pipeline configuration 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>`

The order of handler elements in the `<service_handler_list>` element determines the order that each handler is called at run time. This sequence is significant in preserving the encryption of SOAP messages in a pipeline that is configured for WS-Security. See [Configuring the pipeline for Web Services Security](#).

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](#).

### 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>` pipeline configuration 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>
```

## The <sign\_body> pipeline configuration element

---

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

6.3

Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

6.1

6.2

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

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>](#) element

### 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 algorithms shown in [How it works: Signing SOAP messages](#).
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.

The configuration elements must be used in the sequential order that they are listed.

### Example

```
<sign_body>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>SIGCERT01</certificate_label>
</sign_body>
```

## The <sts\_authentication> pipeline configuration element

---

Specifies that a Security Token Service (STS) must be used for authentication and determines what type of request is sent.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

### Used in:

- Service provider
- Service requester

### Contained by:

- [<dfhwsse\\_configuration>](#) element

### Attributes

#### 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.

**6.3** **Beta**

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

#### 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.

**6.3** **Beta**

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

## 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.

6.3

Beta

Support for SAML using the CICS Security Token Service is removed as of CICS TS 6.3.

## 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 [The `<auth\_token\_type>` element](#).
  - 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.

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

## 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 <sts\_endpoint> pipeline configuration element

Specifies the location of the Security Token Service (STS).

**Stabilized feature:** Support for SAML using the CICS Security Token Service is stabilized. See also [Stabilization notices and discontinued functions](#).

6.3 Support for SAML using the [CICS Security Token Service](#) is removed as of CICS TS 6.3.

6.2 If this function is used, the CICS\_STABILIZED\_FUNCTIONS health check issues message DFHH0955.

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

6.3 Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

6.1 6.2 Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized.

### Used in:

- Service provider
- Service requester

### Contained by:

- [<dfhwsse\\_configuration>](#) element

### 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 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 an IBM 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

### Example 1

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>
```

### Example 1

In this example, the endpoint is configured to use CICS SAML support.

```
<sts_endpoint>
  <endpoint>cics://PROGRAM/DFHSAML</endpoint>
</sts_endpoint>
```

## The `<terminal_handler>` pipeline configuration 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>
```

### Example: Enabling non-Java processing of JSON messages

To enable non-Java processing of JSON messages, specify the terminal handler program as DFHPIJT:

```
<terminal_handler>
  <handler>
    <program>DFHPIJT</program><handler_parameter_list/>
  </handler>
</terminal_handler>
```

**Note:** When you use DFHPIJT as the terminal handler, do not define an application handler in the pipeline configuration file, that is, the pipeline configuration file should not contain an `<apphandler>` element. If an application handler is specified, it is not invoked.

## The `<transport>` pipeline configuration 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>` element
- `<requester_pipeline>` element

### 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

If any of the configuration elements are used, they must be used in the sequential order that they are listed.

### Example

```
<transport>
  <default_transport_handler_list>
    ...
  </default_transport_handler_list>
</transport>
```

## The <transport\_handler\_list> pipeline configuration 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>
```

## The <wsse\_handler> pipeline configuration element

---

### Stabilized feature:

Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. Consider securing SOAP messages using Transport Layer Security as an alternative. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for signing and encrypting SOAP messages for the WS-Security feature is removed as of CICS TS 6.3.

**6.1** **6.2** Support for signing and encrypting SOAP messages for the WS-Security feature is stabilized. 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> element

### Contains:

- A <dfhwsse\_configuration> element

In a provider pipeline configuration file, the CICS supplied message handler for WS-Security might have to decrypt an encrypted 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 `<xop_options>` pipeline configuration 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>` element

### Attributes

#### `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>
```

```
        </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.



---

## Appendix B. Containers used in web services pipelines

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 by 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 header processing programs, see [Header processing programs](#). For information about the header processing program interface, 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 23. Structure of the DFHERROR container

Field name	Length	Contents
PIISNEB-MAJOR-VERSION	1 byte	"1"
PIISNEB-MINOR-VERSION	1 byte	"1"
PIISNEB-ERROR-TYPE	1 byte	A numeric value denoting the type of error. See "Values for the PIISNEB-ERROR-TYPE field" on page 496.
PIISNEB-ERROR-MODE	1 byte	Possible values are as follows: <b>P</b> The error occurred in a provider pipeline. <b>R</b> The error occurred in a requester pipeline. <b>T</b> The error occurred in a Trust client.
PIISNEB-ABCODE	4 bytes	The abend code when the error is associated with a transaction abend.
PIISNEB-ERROR-CONTAINER1	16 bytes	The name of the container when the error is associated with a container.
PIISNEB-ERROR-CONTAINER2	16 bytes	The name of the second container when the error is associated with more than one container.
PIISNEB-ERROR-NODE	8 bytes	The name of the handler program in which the error occurred.

#### Values for the PIISNEB-ERROR-TYPE field

- 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 shown in [Figure 42 on page 497](#).

```

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).

```

*Figure 42. COBOL declaration*

[Table 24 on page 497](#) lists the language copybooks that map the container.

<i>Table 24. Copybooks that map the container</i>	
<b>Language</b>	<b>Copybook</b>
COBOL	DFHPIUCO
PL/I	DFHPIUCL
C and C++	dfhpiuch.h
Assembler	DFHPIUCD

### **DFHFUNCTION container**

Container DFHFUNCTION indicates where in a pipeline a program is being called. See [“DFHFUNCTION container” on page 499](#).

### **DFHHTTPMETHOD container**

Container DFHHTTPMETHOD is available to application programs in all HTTP provider mode CICS pipelines. DFHHTTPMETHOD 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.

DFHHTTPMETHOD is a container of DATATYPE(CHAR) and is 8 characters long.

### **DFHHTTPSTATUS container**

Container DFHHTTPSTATUS is used to specify the HTTP status code and status text for a message produced in the response phase of a service provider pipeline. DFHHTTPSTATUS is a container of DATATYPE(CHAR).

The content of the DFHHTTPSTATUS container must be the same as the initial status line of an HTTP response message, which follows this 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 IBM MQ transport.

If the container contains more than 45 bytes of data, CICS sends 45 bytes and ignores the remaining data.

## **DFHMEDIATYPE container**

Container DFHMEDIATYPE is used to specify the media type for a message produced in the response phase of a service provider pipeline. DFHMEDIATYPE is a container of DATATYPE(CHAR).

The DFHMEDIATYPE container is ignored when the pipeline uses the IBM MQ transport.

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.

### **Examples**

```
text/plain
```

```
image/svg+xml
```

## **DFHNORESPONSE container**

In the request phase of a service requester pipeline, container DFHNORESPONSE indicates that the service provider is not expected to return a response. DFHNORESPONSE is a container of DATATYPE(CHAR).

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**

Container DFHREQUEST contains the request message that is processed in the request phase of a pipeline. DFHREQUEST is a container of DATATYPE(CHAR).

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 DFHFUNCTON 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.

See also:

- [Passing a message to the next message handler in the pipeline](#)
- [Working with messages in a non-terminal message handler](#)

## **DFHRESPONSE container**

Container DFHRESPONSE 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. DFHRESPONSE is a container of DATATYPE(CHAR).

The DFHRESPONSE container is present when a message handler is called and the DFHFUNTION 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 DFHFUNTION 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.

## **DFHWS-NODEJSAPP container**

Container DFHWS-NODEJSAPP holds the name of the NODEJSAPP resource for this pipeline. The container is valid only for a provider mode pipeline, and only if the pipeline is started through a NODEJSAPP resource using the invoke function from the `ibm-cics-api` module. DFHWS-NODEJSAPP is a container of DATATYPE(CHAR).

See also [CICS-supported conversions](#).

## **DFHFUNTION container**

DFHFUNTION 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 DFHFUNTION that occur are RECEIVE-REQUEST, PROCESS-REQUEST, and SEND-RESPONSE.

Figure 43 on page 500 shows the sequence in which the handlers are called and the values of DFHFUNTION that are passed to each handler.

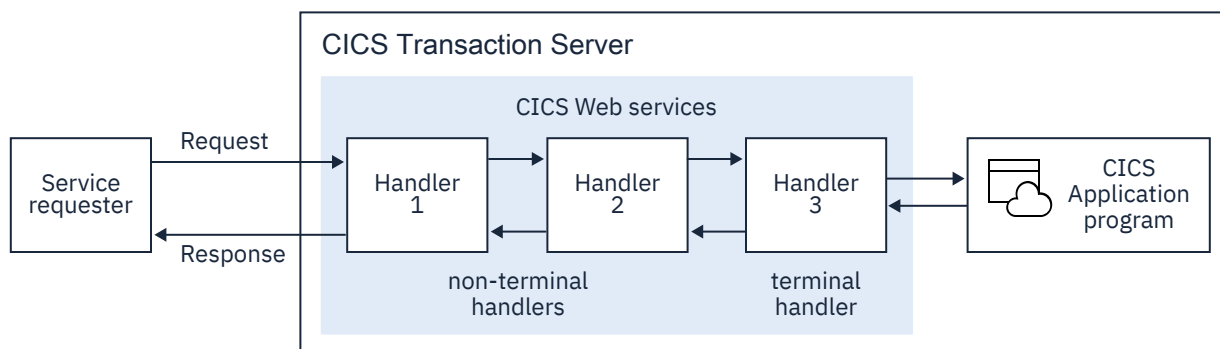


Figure 43. Sequence of handlers in a service provider pipeline

Sequence	Handler	DFHFUNTION
1	Handler 1	RECEIVE-REQUEST
2	Handler 2	RECEIVE-REQUEST
3	Handler 3	PROCESS-REQUEST
4	Handler 2	SEND-RESPONSE
5	Handler 1	SEND-RESPONSE

In a service requester pipeline that sends a request and receives a response, the values of DFHFUNTION that occur are SEND-REQUEST and RECEIVE-RESPONSE. Figure 44 on page 501 shows the sequence in which the handlers are called, and the values of DFHFUNTION that are passed to each handler.

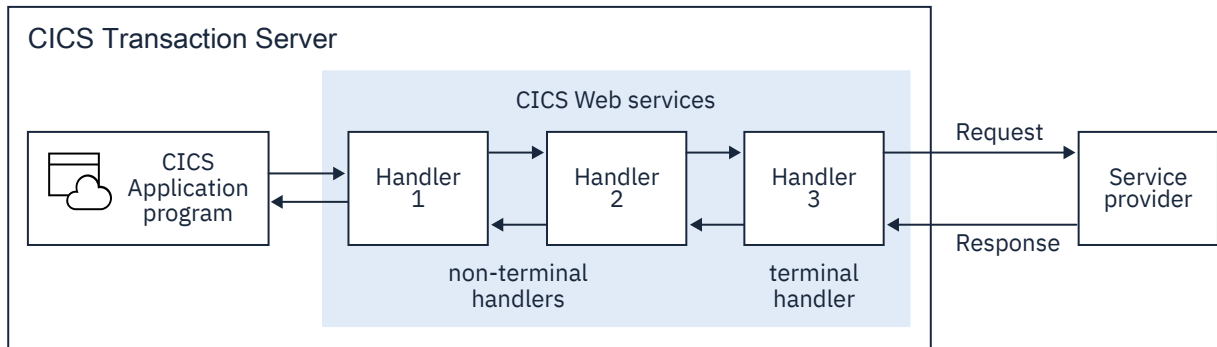


Figure 44. Sequence of handlers in a service requester pipeline

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

## 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; in this case, a context switch occurs causing the program named in the `<apphandler>` element to run in a separate task.

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.

### **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.

### **DFHWS-DATA container**

Container DFHWS-DATA is used in service requester applications and optionally in service provider applications that are deployed with the CICS web services assistant. DFHWS-DATA is a container of DATATYPE(BIT) that 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-DPLTRANID container**

DFHWS-DPLTRANID is a container of DATATYPE(CHAR) that can contain the transaction identifier of the mirror task to be used to run the target application program remotely. If this container is created in a service provider pipeline, it should contain the transaction identifier of the mirror task to be used to run the application in a remote region. If the container is to have any effect, it must be created before control is passed to the target application program. If the container is not present, remote web service application programs are executed under the default mirror transaction, CSMI.

### **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 userabend).
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 the 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 whether CICS expects a response from an external provider. In requester mode, the time that CICS waits for a response is configured with 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 25. 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

Container DFHWS-OPERATION is usually used in a service provider application deployed with the CICS web services assistant. DFHWS-OPERATION is a container of DATATYPE(CHAR) that 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 data is a binary fullword and is one of the following values:

- 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 transaction 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.

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.

For details, see [“Using the DFHWS-URI container” on page 509](#).

### **DFHWS-URI-RESID container**

Container DFHWS-URI-RESID is available only to applications attached by a JSON pipeline. DFHWS-URI-RESID is a container of DATATYPE(CHAR) that 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.

#### **Example**

Assume that the URIMAP that matched the incoming request has a PATH of /JSONServices/ CustomerDetails/\* and that 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 can 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.
- 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**

Container DFHWS-URI-QUERY is available to application programs in all HTTP provider mode CICS pipelines. DFHWS-URI-QUERY is a container of DATATYPE(CHAR) that holds the query string fragment of the URI.

#### **Example**

Assume that 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-URIMAP container**

DFHWS-URIMAP is a container of DATATYPE(CHAR). In a requester pipeline, DFHWS-URIMAP contains the 8-character name of the URIMAP used on the **INVOKE SERVICE** command, or 8 blanks if a URIMAP was not specified. For a provider pipeline, DFHWS-URIMAP contains the 8-character name of the URIMAP that matched the inbound request. You cannot change the contents of this container.

### **DFHWS-URIMAPPATH container**

DFHWS-URIMAPPATH is a container of DATATYPE(CHAR) that 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), and it holds the name of the web service that specifies the execution environment when the target application has been deployed by using the web services assistant.

### **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.

<i>Table 26. Structure of the DFHWS-MTOM-IN container</i>		
<b>Field name</b>	<b>Length (bytes)</b>	<b>Contents</b>
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: <b>0</b> The MTOM message contains binary attachments. <b>1</b> The MTOM message does not contain binary attachments.
XOP_MODE	4	Contains one of the following values: <b>0</b> No XOP processing takes place. <b>1</b> XOP processing takes place in compatibility mode. <b>2</b> 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.

<i>Table 27. Structure of the DFHWS-MTOM-OUT container</i>		
<b>Field name</b>	<b>Length (bytes)</b>	<b>Contents</b>
MTOM_STATUS	4	Indicates whether MTOM is enabled:  <b>0</b> MTOM is not enabled. The outbound message is sent in SOAP format.  <b>1</b> 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:  <b>0</b> Do not send an MTOM message when there are no binary attachments.  <b>1</b> Send an MTOM message when there are no binary attachments.
XOP_MODE	4	Indicates what XOP processing should take place:  <b>0</b> No XOP processing takes place.  <b>1</b> XOP processing takes place in compatibility mode.  <b>2</b> XOP processing takes place in direct mode.

### **DFHWS-WSDL-CTX container**

Container DFHWS-WSDL-CTX is used in either a service provider or a service requester application deployed with the CICS web services assistant. DFHWS-WSDL-CTX is a container of DATATYPE(CHAR) that 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) that 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) that 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.

## Using the 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 is 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.

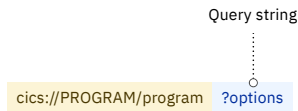
The SOAP message is sent to the service defined by the URI in DFHWS-URI.

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 IBM MQ URI for external services. A service can also use a CICS URI for a service that is provided by another CICS application:

### URI format 1

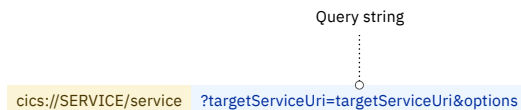
`cics://PROGRAM/program?options`



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 format 2

`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.

### URI format 3

`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	Description
PROGRAM	<b>userid</b>	Optional
PROGRAM	<b>transid</b>	Optional
PROGRAM	<b>maxCommareaLength</b>	Optional
PROGRAM	<b>newTask</b>	Optional. Must be yes or not specified at all if you specify <b>userid</b> or <b>transid</b> .
PROGRAM	<b>targetServiceUri</b>	Not supported
SERVICE	<b>userid</b>	Optional
SERVICE	<b>transid</b>	Optional
SERVICE	<b>maxCommareaLength</b>	Not supported
SERVICE	<b>newTask</b>	Optional. Must be yes or not specified at all if you specify <b>userid</b> or <b>transid</b> .
SERVICE	<b>targetServiceUri</b>	Required
PIPELINE	<b>userid</b>	Not supported
PIPELINE	<b>transid</b>	Not supported
PIPELINE	<b>maxCommareaLength</b>	Not supported
PIPELINE	<b>newTask</b>	Not supported
PIPELINE	<b>targetServiceUri</b>	Required

**Examples of CICS URIs****Example 1**

In this 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.

## Example 2

In this 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.

## Example 3

In this 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.

## The header processing program containers

---

The CICS-provided SOAP 1.1 and SOAP 1.2 message handlers link to header processing programs by using channel DFHHHC-V1. The containers that are passed on the channel include DFHHEADER, which is specific to the header processing program interface, and other containers such as DFHHEADER, DFHWS-XMLNS, DFHWS-BODY, and DFHXMLSS-PARSE, which are available elsewhere in your pipeline but have specific uses in a header processing program.

The header processing program interface also passes sets of control containers, context containers and user containers that are accessible to all the header processing programs and message handler programs in the pipeline. For information about these containers, see [Appendix B, “Containers used in web services pipelines,”](#) on page 495.

### DFHHEADER container

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.

## DFHWS-XMLNS container

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, as in the following example.

### 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.

## DFHWS-BODY container

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, because every SOAP message must contain a <Body> element.

## DFHXMLSS-PARSE container

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.

## 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. Use this container only with channel DFHWSTC-V1 for the Trust client interface.

The token must be in XML format.

## 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
<code>http://schemas.xmlsoap.org/ws/2005/02/trust/status/valid</code>	The security token is valid.
<code>http://schemas.xmlsoap.org/ws/2005/02/trust/status/invalid</code>	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
<code>http://schemas.xmlsoap.org/ws/2005/02/trust/Issue</code>	The STS issues a token in exchange for the one that is sent in the DFHWS-IDTOKEN container.
<code>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</code>	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-STREASON container

DFHWS-STREASON 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](#).

## DFHWS-STSURE container

DFHWS-STSURE 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

---

This reference describes the read-only containers that are used by CICS SAML support. In this reference information, *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.

**Stabilized feature:** Support for SAML using the CICS Security Token Service is stabilized. See also [Stabilization notices and discontinued functions](#).

**6.3** Support for SAML using the [CICS Security Token Service](#) is removed as of CICS TS 6.3.

**6.2** If this function is used, the CICS\_STABILIZED\_FUNCTIONS health check issues message DFHH0955.

## DFHSAML-AnnnVmmm container

This 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-ATTR*nnn*.

## DFHSAML-ASSQNAME container

DFHSAML-ASSQNAME is a container of DATATYPE(CHAR). It contains the SAML Assertion namespace. Possible values are as follows:

### 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-ATTR*nnn* 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-ATTRF*nnn* container**

DFHSAML-ATTRF*nnn* 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-ATTRN*nnn* container**

DFHSAML-ATTRN*nnn* 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-ATTRS*nnn* container**

DFHSAML-ATTRS*nnn* 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-ATTRY*nnn* container**

DFHSAML-ATTRY*nnn* 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-AUDNR*nnn* container**

DFHSAML-AUDNR*nnn* 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-PROXY $nnn$ container**

DFHSAML-PROXY $nnn$  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-*nnnnnnn***

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.

---

## Appendix C. JSON transformer linkable interface containers

The JSON transformer linkable interface is a CICS-supplied program that can be called to perform data transformations to and from JSON. It can create JSON data from structured application data, or structured application data from JSON data.

You can use the linkable interface to transform JSON from any source, whether over a protocol such as IBM MQ or from a JSON database, as compared to the JSON web services support, which is HTTP and provider-mode only. For more information about JSON, see [Concepts of JSON web services](#).

To use the JSON transformer linkable interface the application program must LINK to the CICS supplied DFHJSON program, with a channel. Containers are used to pass input data to DFHJSON, and to receive output data from DFHJSON, the details of which are provided in subtopics.

CICS provides two mechanisms for transforming structured application data to and from JSON data, one using a JVM server, the other without. The JVM server approach experiences the advantages of the Java platform (such as eligibility for IBM Z Application Assist Processor (zAAP) where available), the non Java approach avoids the need to configure a JVM server. The application selects which is used by using the DFHJSON\_JVMSEVR container. The contents identify the JVM server in which to transform the data, the absence of this container indicates that the non Java transformation service is used.

To use this service, you must have a suitable JSONTRANSFRM bundle part that is installed in CICS. These are generated by using the JSON assistants. For more information, see [Mapping and transforming application data and JSON](#).

### DFHJSON-JSON container

- When you transform application data to JSON: This container contains the JSON created by the transformer corresponding to the application data input.
- When you transform JSON to application data: This container contains the JSON provided by the user application to be transformed.

### DFHJSON-DATA container

DFHJSON-DATA is a container of DATATYPE(BIT).

- When you transform application data to JSON: This container contains the application data that is provided by the user to be transformed into JSON.
- When you transform JSON to application data: This container contains the application data that is created by the transformed data corresponding to the JSON input.

### DFHJSON-TRANSFRM container

DFHJSON-TRANSFRM is a container of DATATYPE(CHAR). It contains the 16 character name of the JSONTRANSFRM bundle part whose mappings are applied to transform the data.

The JSONTRANSFRM is a bundle part. It is contained in a bundle that is generated by using the JSON assistant.

For more information about how to view bundle parts, see [Bundle Parts view in the CICS Explorer product documentation](#).

Alternatively, you can use the **INQUIRE BUNDLEPART** command. See [INQUIRE BUNDLEPART](#).

## **DFHJSON-JVMSEVR container**

DFHJSON-JVMSEVR is a container of DATATYPE(CHAR). It contains the eight character name of the JVMSEVR in which the target program is to run.

If the container is not provided, or if the container is present but with a length of zero, then data transformation will be carried out within CICS rather than in a JVM server. If you want to carry out transformation in the CICS supplied Axis2 JVM server, then specify the value DFHAXIS in this container.

## **DFHJSON-ERROR container**

DFHJSON-ERROR is a container of DATATYPE(BIT). Errors that are generated during validation or execution of DFHJSON are returned as a fullword binary value in the DFHJSON-ERROR container. The following errors can be returned by DFHJSON:

- 1** The DFHJSON-TRANSFRM container was not present.
- 2** The name of the JSONTRANSFRM provided in the DFHJSON-TRANSFRM container was longer than 16 characters.
- 3** A container had the wrong data type. DFHJSON-ERRORMSG contains the name of the incorrect container.
- 4** The name of the JVMSEVR provided in the DFHJSON-JVMSEVR container was longer than eight characters.
- 11** The JSONTRANSFRM named in DFHJSON-TRANSFRM container was not found.
- 12** The JSONTRANSFRM named in DFHJSON-TRANSFRM container was not in the ENABLED state.
- 13** Neither the DFHJSON-DATA or the DFHJSON-JSON container is present, one of these containers must be specified.
- 14** Both the DFHJSON-DATA and the DFHJSON-JSON containers were present, only one of these containers must be provided.
- 15** A data transformation error occurred. DFHJSON-ERRORMSG contains details of the error.
- 16** An error occurred while parsing JSON provided in the DFHJSON-JSON container. DFHJSON-ERRORMSG contains details of the error.
- 17** An error occurred in the Java portion of the transformer. Contact IBM support for assistance.
- 20** An unexpected error occurred. Contact IBM support for assistance.
- 51** The JVM server that is named in the DFHJSON-JVMSEVR container was not found.
- 52** The JVM server that is named in the DFHJSON-JVMSEVR container was not in the ENABLED state.

## **DFHJSON-ERRORMSG container**

DFHJSON-ERRORMSG is a container of DATATYPE(CHAR). This container is created if an error occurs during processing. It contains a human-readable extra error message for example, the error message from the JSON parser. This message is only populated if extra details are needed.

For more information about error codes, see [“DFHJSON-ERROR container” on page 520](#).



---

## Appendix D. Web services samples

CICS supplies a set of samples that you can use as a starting point for developing applications and configuring CICS.

The samples are categorized as follows:

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

- [“The CICS catalog manager example application” on page 523](#)

### JSON samples

Use these examples to help you understand JSON requests.

- [“JSON samples” on page 558](#)

---

## 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 could be used to install and deploy the application. The CICS Explorer is an Eclipse-based graphical tooling interface for CICS.

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

### 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 45 on page 524](#) shows the structure of the base application.

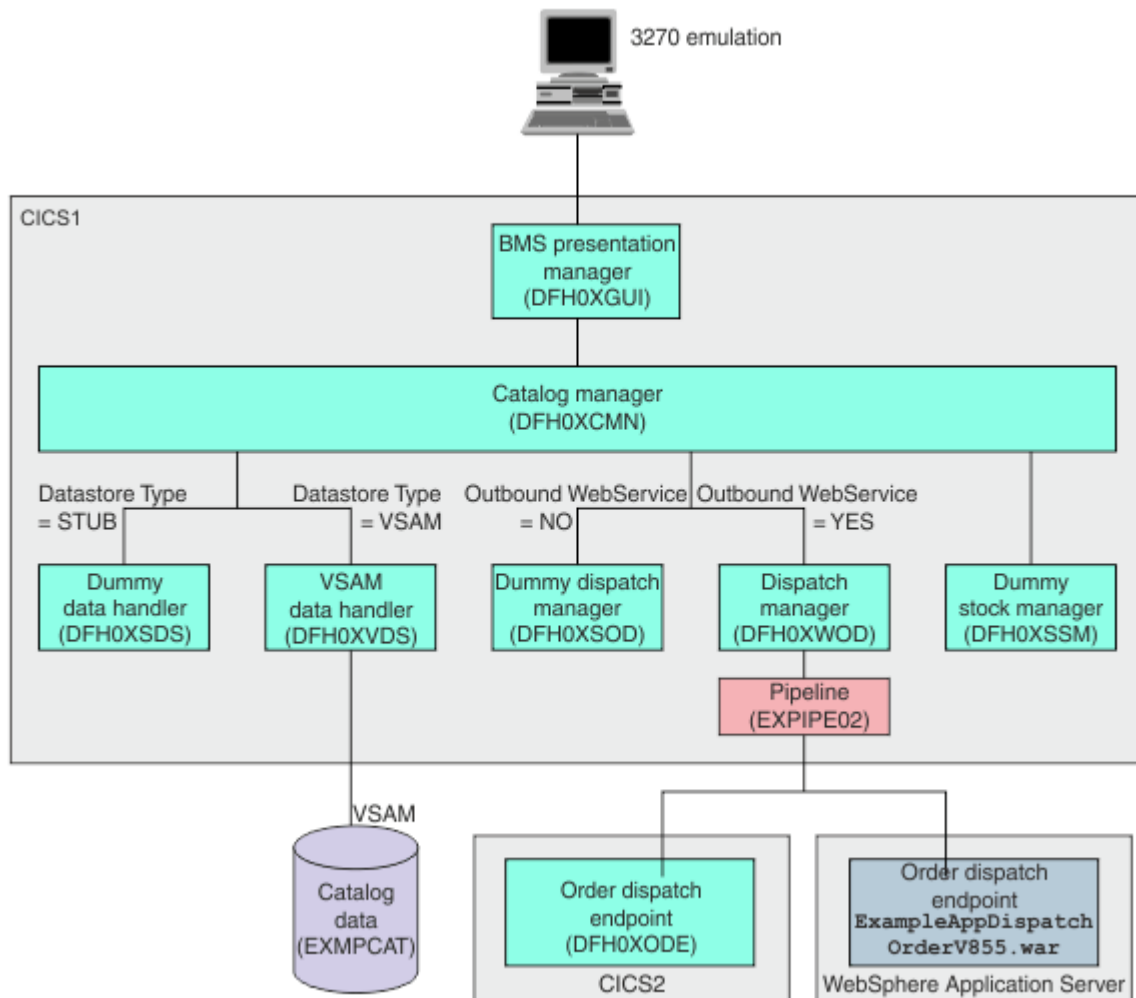


Figure 45. Structure of the base application

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: DFH0XWOD is a web service requester that invokes a remote order dispatch end point, and DFH0XSOD 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; ExampleAppDispatchOrderV855.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. Alternatively, CEDA can be used. See details below.

- 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. Alternatively, CEDA can be used. See details below.
- 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.

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS. The file definitions are in group DFH\$EXBS. Copy group DFH\$EXBS into a new group with a name not starting with DFH. Edit the file definitions in the new group to specify the correct file names in the DSNAME parameter, then install the new group.

## 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.

## About this task

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

## 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.

### EGUI

- 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.
- b. Select **New > Transaction Definition**.
- c. Type EGUI in the **Name** text box, specify a name in the **Resource/CSD Group** text box, and DFH0XGUI in the **Program Name** text box.
- d. Click **Finish** to create the EGUI transaction definition.

Use the default values for all other attributes.

### ECFG

- a. Copy the definitions for transaction ECFG from the CICS supplied group DFH\$EXBS by right-clicking DFH\$EXBS in the **Resource Group Definitions** window.
- b. Select **New > Transaction Definition**.
- c. Type ECFG in the **Name** text box, specify a name in the **Resource/CSD Group** text box, and DFH0XCFG in the **Program Name** text box.
- d. Click **Finish** to create the ECFG transaction definition.

Use the default values for all other attributes.

### ECLI (optional)

- a. Copy the definitions for transactions ECLI from the CICS supplied group DFH\$EXWS by right-clicking DFH\$EXWS in the **Resource Group Definitions** view.
- b. Select **New > Transaction Definition**.
- c. Type ECLI in the **Name** text box, specify a name in the **Resource/CSD Group** text box, and DFH0XCUI in the **Program Name** text box.
- d. 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 548](#).

- b) Copy the PROGRAM resource definitions for the following COBOL programs.

<b>Member name</b>	<b>Description</b>
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.

Table 28. SDFHSAMP members containing COBOL source for the base application (continued)	
Member name	Description
DFHOXODE	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.
DFHOXSDS	A <i>stubbed</i> or dummy version of the data store program that allows the application to work when the VSAM catalog file has not been set up. DFHOXSDS uses data defined in the program rather than data stored in a VSAM file.
DFHOXSOD	A stubbed version of the order dispatch program. It sets the return code in the COMMAREA to 0 and returns to its caller. DFHOXSOD is used when outbound web services are not required.
DFHOXSSM	A stubbed version of the stock manager (replenishment) program. DFHOXSSM sets the return code in the COMMAREA to 0 and returns to its caller.
DFHOXVDS	The VSAM version of the data store program. DFHOXVDS accesses the VSAM file to perform reads and updates of the catalog.
DFHOXWOD	The web service version of the order dispatch program. DFHOXWOD 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 DFHOXCUI 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$EXWS)
ENTER COMMANDS
NAME      TYPE      GROUP
DFHOXCUI PROGRAM  DFH$EXWS
ECLI     TRANSACTION DFH$EXWS
EXMPPORT TCPIP SERVICE DFH$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.

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

## 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. You can use the CEOT transaction to set upper case translation off for your terminal session.

### 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 DFHOXCMN.

**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 DFHOXSDS.

**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 DFHOXVDS.

**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 DFHOXSOD.

**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 DFHOXWOD.

**Stock Manager**

Specify the name of the stock manager program. The program supplied with the example application is DFHOXSSM.

**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

- If there is sufficient stock to fulfill the order, enter the following information:
  - Complete the **Order Quantity** field.  
Specify the number of items you want to order.
  - Complete the **User Name** field.  
Enter a 1-to 8-character string. The base application does not check the value that is entered here.
  - Complete the **Charge Dept** field.  
Enter a 1-to 8-character string. The base application does not check the value that is entered here.
- Press Enter to submit the order and return to the main menu.
- Press F3 to end the applications.

## Web service support for the example application

The web service support extends the example application, providing two Java versions of a web server front end client, and a Java and COBOL version of the web service endpoint for the order dispatcher component.

The two versions of the web client front end and one version of the web service endpoint are supplied as Java web archive files (WARs) that run in the Java EE 6 web profile environment provided by the latest version of WebSphere Application Server or the latest CICS TS Liberty JVM server. The second version of the web service endpoint is supplied as a CICS service provider application program (DFH0XODE).

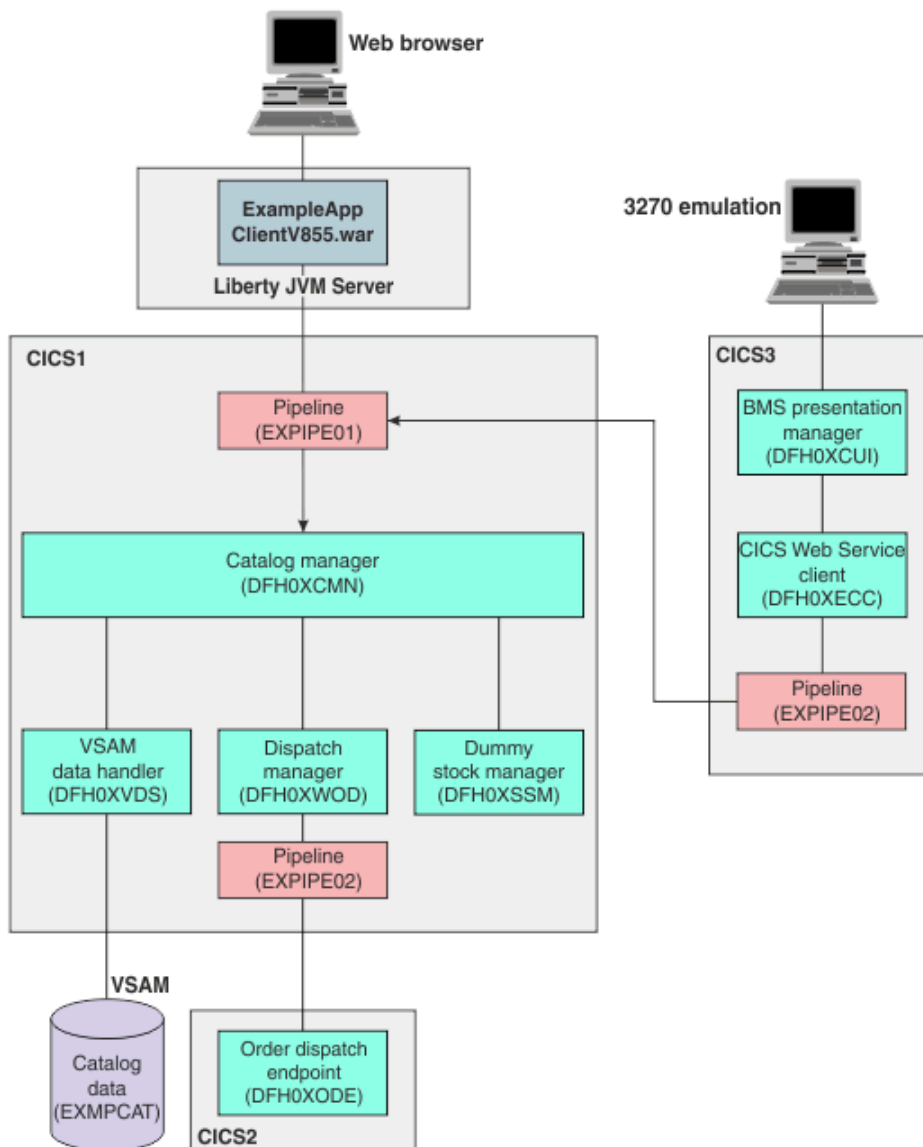
File	Description
ExampleAppClientV855.war	Web service front end client to catalog manager
ExampleAppWrapperClientV855.war	Web service front end client to web service wrappers
ExampleAppDispatchOrderV855.war	Order dispatch web service provider application

These WARs were exported from Dynamic Web Projects. For information about deploying the WAR files, see [Deploying applications to a JVM server](#).

You must enable the `jax-ws` feature in the Liberty JVM server, for example by adding the following to the Liberty server configuration file, `server.xml`:

```
<feature>jaxws-2.2</feature>
```

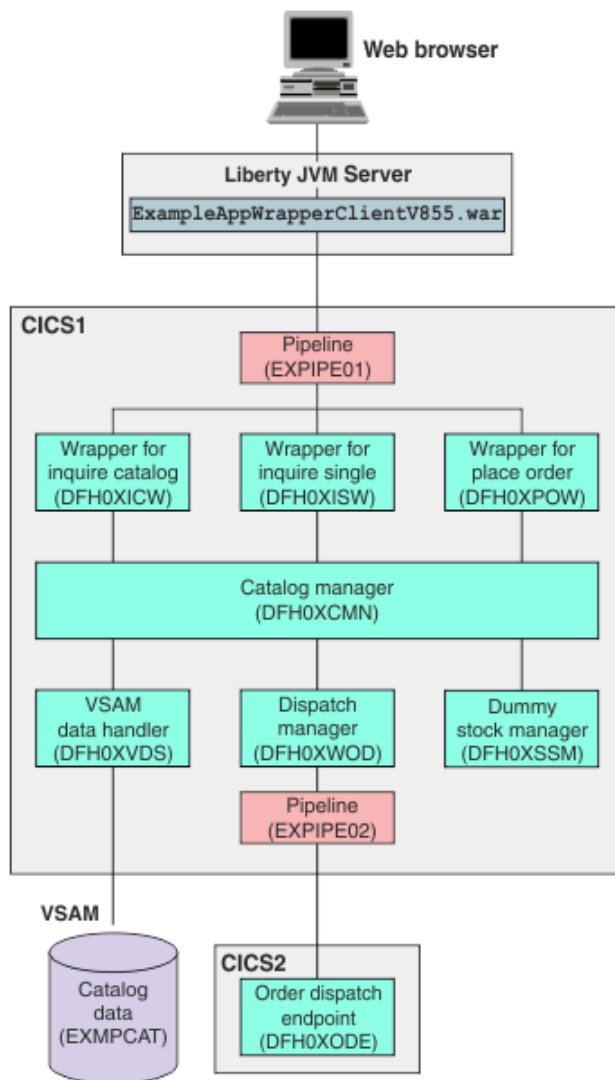
Figure 1 shows a configuration of the example application with one version of the web client front end and CICS service provider as the order dispatch web service end point. It also includes a web service client on a CICS system.



In this configuration, the application is accessed through two different clients:

- A web browser client connected to Liberty JVM server, in which ExampleAppClientV855.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.



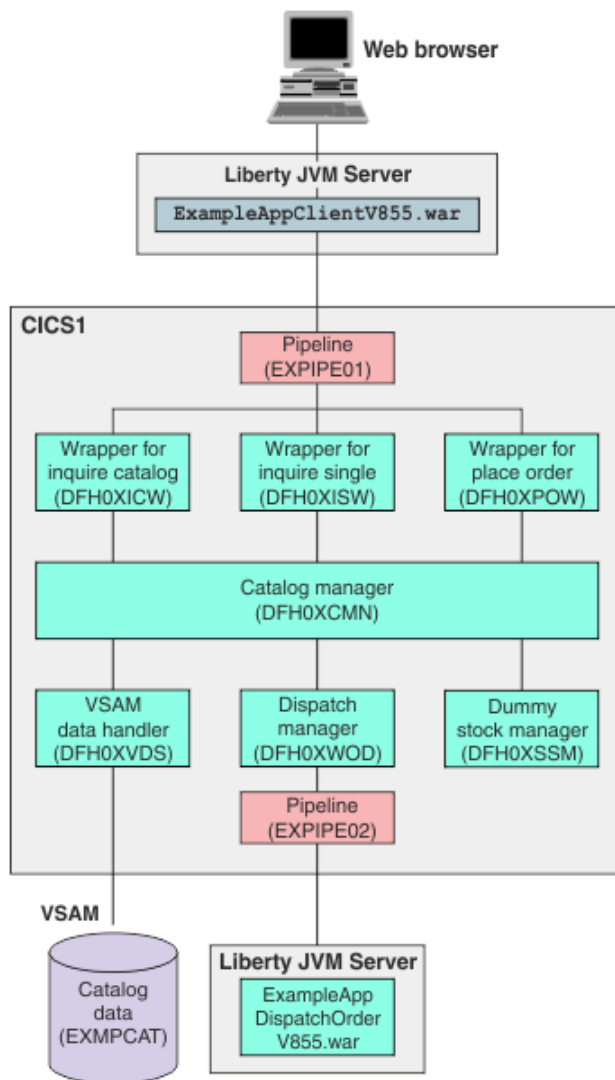
In this configuration, the web browser client is connected to Liberty JVM server, in which `ExampleAppWrapperClientV855.war` is deployed. In CICS, three wrapper applications (for the inquire catalog, inquire single, and place order functions) are deployed as service provider applications. They in turn 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 using the ECFG configuration transaction:

- Outbound WebService? to YES
- Outbound WebService URI to the URI where the Dispatch Order end point is being deployed on, e.g. `http://cics2:8080/exampleApp/dispatchOrder`

For further details on setting up the example application, see [Configuring the example application](#).

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 ExampleAppClientV855.war is deployed. The order dispatch web service end point ExampleAppDispatcherV855.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 using the ECFG configuration transaction

- OutboundWebService? to YES
- OutboundWebService URI to the URI where the Dispatch Order end point is being deployed on, e.g. `http://mylibertyserver:9080/ExampleAppDispatcherV855/DispatchOrder`

For further details on setting up the example application, see [Configuring the example application](#).

## Configuring code page support

As supplied, the example application uses two coded character sets. You must configure your system to enable data conversion between the two character sets.

### 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 [Unicode data conversion by z/OS](#).

### 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.

#### About this task

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

### Procedure

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 a CSD group in the **CSD Group** text box, and type the program name in the **Name** text box. Click **Finish** to define the PROGRAM definition.

Create definitions for the following COBOL programs:

<b>Member name</b>	<b>Description</b>
DFH0XECC	Web services client program
DFH0XICW	Wrapper program for the inquireCatalog service.
DFH0XISW	Wrapper program for the inquireSingle service.
DFH0XPOW	Wrapper program for the purchaseOrder service.

### 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

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

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 `basicsoap11java requester.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.

- a) 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.
- b) 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
- c) Use the default values for all other attributes.

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

## 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.

### About this task

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

### 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 538 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	EXINQSW	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
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

- Use the CICS Explorer to create a URIMAP definition for the inquire catalog function of the example application.
  - Create a URIMAP definition in the CICS Explorer by selecting **Definitions > URI Map Definition**.
  - Right-click in the **URI Map Definitions** view and select **New** to create a new URIMAP definition.
  - 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.
  - Click **Finish** to finish the URIMAP definition.
  - 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 SOAPPRT.
- 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

Function	URIMAP name
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.

Optional: Those users that do not use CICS Explorer can use the CEDA transaction to amend and install the resources contained in groups DFH\$EXBS and DFH\$EXWS.

## 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 ExampleAppClientV855.war and ExampleAppWrapperClientV855.war:

- CICS Liberty JVM server with the latest WebSphere Liberty

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 Liberty JVM server on which the web client is installed.
  - For ExampleAppClientV855.war, use URL `http://mylibertyserver:9080/ExampleAppClientV855/`
  - For ExampleAppWrapperClientV855.war, use URL `http://mylibertyserver:9080/ExampleAppWrapperClientV855/`
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 TCPIP SERVICE 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://mylibertyserver:9080/ExampleAppClientV855/`, where *mylibertyserver* is the host name of the server on which the web service client is installed.
2. Click the **INQUIRE** button.
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 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.
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 confirms 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 (DFHOXCMN) is deployed as a web service. The interface to this program is a COMMAREA; the structure of the COMMAREA is defined in the copybook DFHOXCP1:

```
* 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/wsd1/inquireSingle.wsd1
*/
```

#### **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/wsd1/inquireSingle.wsd1**

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
                </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 TCPIP SERVICE. -->
    <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&connectionFactory=()&targetService=/exampleApp/
    inquireSingles.log&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/wsbinding/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.

<i>Table 30. SDFHSAMP members containing BMS maps</i>	
<b>Member name</b>	<b>Description</b>
DFH0XS1	BMS macros for the mapset consisting of the map (EXMENU) for the <b>Main Menu</b> screen and the map (EXORDR) for the <b>Details of your order</b> screen.
DFH0XS2	BMS macros for the mapset consisting of the map (EXINQC) for the <b>Inquire Catalog</b> screen.
DFH0XS3	BMS macros for the mapset consisting of the map (EXCONF) for the <b>Configure CICS example catalog application</b> 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

<i>Table 31. SDFHSAMP members containing COBOL source for the base application</i>	
<b>Member name</b>	<b>Description</b>
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.

Table 31. SDFHSAMP members containing COBOL source for the base application (continued)

Member name	Description
DFH0XSDS	A <i>stubbed</i> or dummy version of the data store program that allows the application to work when the VSAM catalog file has not been set up. DFH0XSDS uses data defined in the program rather than data stored in a VSAM file.
DFH0XSOD	A <i>stubbed</i> 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 <i>stubbed</i> 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 32. 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 to produce <code>inquireCatalog.wsdl</code> and <code>inquireCatalog.wsbinding</code> .
DFH0XCP4	Defines a data structure for an inquire single request and response. Used as input to DFHLS2WS to produce <code>inquireSingle.wsdl</code> and <code>inquireSingle.wsbinding</code> .
DFH0XCP5	Defines a data structure for a place order request and response. Used as input to DFHLS2WS to produce <code>placeOrder.wsdl</code> and <code>placeOrder.wsbinding</code> .
DFH0XCP6	Defines a data structure for a dispatch order request and response. Used as input to DFHLS2WS to produce <code>dispatchOrder.wsdl</code> and <code>dispatchOrder.wsbinding</code> .
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 33. SDFHSAMP members containing COBOL source code for the web service client application that runs in CICS

Member name	Description
DFH0XCUI	Program invoked by transaction ECLI to manage the sending of the BMS maps to the terminal user and the receiving of the maps from the terminal user. It links to program DFH0XECC.
DFH0XECC	Makes outbound web service requests to the base application, using the EXEC CICS INVOKE WEBSERVICE command. The web service specified is one of the following:  <pre> inquireCatalogClient inquireSingleClient placeOrderClient </pre>

Table 34. SDFHSAMP members containing COBOL copybooks for the web service client application that runs in CICS. They are all generated by DFHWS2LS, and are included by program DFH0XECC.

Member name	Description
DFH0XCPA	Defines the data structure for the inquire catalog request.
DFH0XCPB	Defines the data structure for the inquire catalog response.
DFH0XCPC	Defines the data structure for the inquire single request.
DFH0XCPD	Defines the data structure for the inquire single response.
DFH0XCPE	Defines the data structure for the place order request.
DFH0XCPF	Defines the data structure for the place order response.

Table 35. SDFHSAMP members containing COBOL source code for the wrapper modules

Member name	Description
DFH0XECC	Web services client program
DFH0XICW	Wrapper program for the inquireCatalog service.
DFH0XISW	Wrapper program for the inquireSingle service.
DFH0XPOW	Wrapper program for the purchaseOrder service.

Table 36. 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.

Table 36. SDFHSAMP members containing COBOL copybooks for the wrapper modules (continued)

Member name	Description
DFH0XWC6	Defines the data structure for the place order response. Program DFH0XPOW includes this copybook

Table 37. 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 38. 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 38. Catalog manager return codes (continued)

Type	Code	Explanation
Remote web service	30	The <b>EXEC CICS INVOKE WEBSERVICE</b> command returned an INVREQ condition
	31	The <b>EXEC CICS INVOKE WEBSERVICE</b> command returned an NOTFND condition
	32	The <b>EXEC CICS INVOKE WEBSERVICE</b> 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	DFHOXCMN 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 01DSP0.

**CA-ORD-USERID**

An 8-character user ID which the application uses for dispatch and billing.

**CA-ORD-CHARGE-DEPT**

An 8-character department ID which the application uses for dispatch and billing.

**CA-ORD-ITEM-REF-NUMBER**

The reference number of the item to be ordered.

**CA-ORD-QUANTITY-REQ**

The number of items required.

**Output parameters****CA-ORD-RETURN-CODE**

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 01STK0.

**CA-STK-ITEM-REF-NUMBER**

The reference number of the item to be ordered.

**CA-STK-QUANTITY-REQ**

The number of items required.

**Output parameters****CA-ORD-RETURN-CODE**

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:

<b>Name</b>	<b>COBOL data type</b>	<b>Description</b>
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

Name	COBOL data type	Description
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.

<i>Table 39. General information record</i>		
Name	COBOL data type	Description
PROGS-KEY	PIC X(9)	Key field for the general information record, containing EXMP - CONF.
filler	PIC X	
DATASTORE	PIC X(4)	A character string that specifies the type of data store program to be used. Values are:  STUB VSAM
filler	PIC X	
DO-OUTBOUND-WS	PIC X	A character that specifies whether the dispatch manager is make an outbound web service request. Values are:  Y N
filler	PIC X	
CATMAN-PROG	PIC X(8)	The name of the catalog manager program.
filler	PIC X	
DSSTUB-PROG	PIC X(8)	The name of the dummy data handler program.
filler	PIC X	
DSVSAM-PROG	PIC X(8)	The name of the VSAM data handler program.
filler	PIC X	
ODSTUB-PROG	PIC X(8)	The name of the dummy order dispatcher module.
filler	PIC X	
ODWEBS-PROG	PIC X(8)	The name of the outbound web service order dispatcher program.

*Table 39. General information record (continued)*

<b>Name</b>	<b>COBOL data type</b>	<b>Description</b>
filler	PIC X	
STKMAN-PROG	PIC X(8)	The name of the stock manager program.
filler	PIC X(10)	

*Table 40. Outbound URL record*

<b>Name</b>	<b>COBOL data type</b>	<b>Description</b>
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 41. Catalog file information record*

<b>Name</b>	<b>COBOL data type</b>	<b>Description</b>
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 42. Server information record*

<b>Name</b>	<b>COBOL data type</b>	<b>Description</b>
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.

## JSON samples

---

Use these examples to help you understand JSON requests.

### Example HTTP GET request using a query string

This is an example of a HTTP GET request using a query string.

```
GET /genapp/customers?name=Joe%20Bloggs/1
Host: www.example.com
```

Where **1** is the query string.

### Example HTTP request with a JSON body

This is an example of a HTTP request with a JSON body.

```
POST /genapp/customers/
Host: www.example.com
Content-Type: application/json
Content-Length: nn 1
```

```
{
  "customers":
  {
    "firstName": "Joe",
    "lastName": "Bloggs",
    "fullAddress":
    {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": 10021
    }
  }
}
```

Where the Content-Length: *nn* **1** is the length of your request.

The COBOL language structure mapping for this example would be as follows:

```
01 CUSTOMERS.
   03 firstname pic x(8).
   03 lastname  pic x(8).
   03 fulladdress.
       05 streetaddress pic x(20).
       05 city          pic x(20).
       05 state         pic xx.
       05 postalcode   pic 9(5).
```

## Notices

---

This information was developed for products and services offered in the United States of America. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 rights 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 grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

For license inquiries regarding double-byte character set (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*

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 jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information 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.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119 Armonk,  
NY 10504-1785  
United States of America*

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 Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## Beta content

Content described as “beta” relates to a release of the program prior to it being made commercially available that may still be under development and therefore, potentially unreliable. Beta content may change or be removed, is not intended for production use, and is provided as-is, without liability, warranty or support, to the full extent permitted by applicable law.

Statements by IBM regarding its plans, directions, and intent are subject to change or withdrawal without notice at the sole discretion of IBM. Information regarding potential future products is intended to outline general product direction and should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for IBM products remain at the sole discretion of IBM.

## Licensed Program Specifications (LPS)

### Where can I find license terms for Monthly License Charge (MLC)?

For more information about the license terms for MLC, see:

- [6.3 IBM CICS Transaction Server Licensed Program Specifications 6.3](#)
- [6.2 IBM CICS Transaction Server Licensed Program Specifications 6.2](#)
- [6.1 IBM CICS Transaction Server Licensed Program Specifications 6.1](#)

### Where can I find license terms for Value Unit Edition (VUE)?

For more information about the license terms for VUE, see:

- [6.3 License Information terms and conditions for Value Unit Edition 6.3](#)
- [6.2 License Information terms and conditions for Value Unit Edition 6.2](#)
- [6.1 License Information terms and conditions for Value Unit Edition 6.1](#)

## Programming interface information

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 6 are included in the following sections of the online product documentation:

- [Developing applications](#)
- [Developing system programs](#)
- [Securing CICS](#)
- [Developing for external interfaces](#)
- [Application development reference](#)
- [Reference: system programming](#)
- [Reference: connectivity](#)

Information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 6, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- [Troubleshooting and support](#)
- [CICS TS diagnostics reference](#)

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 6 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services
- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSplex System Manager (CICSplex SM) Managing Workloads
- CICSplex SM Managing Resource Usage
- CICSplex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 6, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSplex SM Problem Determination Guide

## Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)<sup>®</sup> 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](http://www.ibm.com/legal/copytrade.shtml).

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat<sup>®</sup>, and Hibernate<sup>®</sup> are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe<sup>™</sup>, the Zowe logo and the Open Mainframe Project<sup>™</sup> are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Red Hat, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

## Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

### Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

### Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications,

or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

## **Rights**

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## **IBM online privacy statement**

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 (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth here:

### **For the CICSplex SM Web User Interface (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 PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

### **For the CICSplex SM Web User Interface (data interface):**

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

### **For the CICSplex SM Web User Interface ("hello world" page):**

Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. 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 PII 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 Privacy Policy](#) and [IBM Online Privacy Statement](#), the section entitled *Cookies, Web Beacons and Other Technologies* and the [IBM Software Products and Software-as-a-Service Privacy Statement](#).



---

# Index

## A

application  
    transform to XML [311](#)  
assistant, JSON [197](#)  
assistant, web services [140](#)  
assistant, XML [311](#)  
Atom feed  
    XML binding  
        creating [388](#)

## B

batch utility  
    JSON assistant [197](#)  
    web services assistant [140](#)  
    XML assistant [311](#)  
binding file  
    XML binding [388](#)  
body, SOAP [20](#)

## C

C and C++  
    mapping to JSON Schema [272](#), [275](#)  
    mapping to XML Schema [349](#), [353](#)  
call web service from Java [194](#)  
catalog sample [523](#)  
channel description [134](#), [176](#)  
channel interface [134](#), [176](#)  
CICS web services  
    Unicode [300](#), [382](#)  
    UTF-16 [300](#), [382](#)  
COBOL  
    mapping to JSON schema [264](#)  
    mapping to JSON Schema [257](#)  
    mapping to XML schema [343](#)  
    mapping to XML Schema [337](#)  
    variable repeating content [385](#)  
command line [451](#)  
command-line [451](#)  
compatibility mode [90](#)  
containers  
    channel description [134](#), [176](#)  
Creating a service requester application [137](#)  
custom security handler [436](#)  
customizing pipeline processing [122](#)

## D

default EPR  
    WSDL 1.1 [105](#)  
DFHJS2LS  
    cataloged procedure [210](#), [222](#), [243](#)  
DFHLS2SC, CICS XML assistant program [388](#)  
DFHSC2LS

DFHSC2LS (*continued*)  
    cataloged procedure [320](#)  
DFHWS2LS  
    cataloged procedure [154](#)  
diagnosing problems  
    service requester [442](#)  
direct mode [90](#)

## E

end point reference  
    default [105](#)  
endpoint reference (EPR) [98](#)  
envelope, SOAP [20](#)  
error [451](#)  
example [523](#)  
EXEC CICS SOAPFAULT CREATE command [179](#)

## F

fault, SOAP [20](#)

## G

Generating mappings from a JSON schema [302](#)  
Generating mappings from language structure [301](#)  
global user exits [122](#)  
GLUEs [122](#)

## H

header, SOAP [20](#)

## I

invoking the trust client [435](#)

## J

Java  
    call web service [194](#)  
Java-based SOAP pipelines [101](#)  
JSON  
    converting to language structure [243](#)  
JSON assistant [197](#), [451](#)  
JSON samples [558](#)  
JSON schema [128](#), [264](#)  
JSON Schema [257](#), [272](#), [275](#), [282](#), [287](#)  
JVM server [194](#)

## M

mapping language structures to XML [388](#)  
mapping to C and C++ [272](#), [275](#), [349](#), [353](#)  
mapping to COBOL [257](#), [264](#), [337](#), [343](#)

mapping to PL/I [282](#), [287](#), [358](#), [362](#)  
mapping XML schemas to application data [390](#)  
maxOccurs  
    in JSON schema [296](#)  
    in XML schema [368](#)  
message handler  
    invoking trust client [435](#)  
minOccurs  
    in JSON schema [296](#)  
    in XML schema [368](#)

## P

parsing XML [384](#)  
persistent message support [41](#)  
pipeline processing  
    customizing [122](#)  
PL/I  
    mapping to JSON Schema [282](#), [287](#)  
    mapping to XML Schema [358](#), [362](#)

## Q

querying XML [383](#)

## R

repeating content [385](#)  
return code [451](#)

## S

sample [523](#)  
schema  
    channel description [134](#), [176](#)  
security handler  
    writing your own [436](#)  
service provider application  
    creating from a data structure [129](#), [173](#)  
service requester  
    diagnosing problems [442](#)  
SOAP  
    body [20](#)  
    envelope [20](#)  
    fault [20](#)  
    header [20](#)  
    overview [17](#)  
    overview of SOAP [17](#)  
SOAP faults [179](#)  
SOAP message  
    example [20](#)  
    structure [20](#)  
SOAP message path [10](#)  
SOAP messages  
    validating against XML Schema [191](#)  
    XML Schema  
        validating SOAP message [191](#)

## T

TRANSFORM DATATOXML [391](#)  
TRANSFORM XMLTODATA [392](#)  
Transforming application data to JSON [304](#)

transforming binary data to XML [391](#)  
Transforming JSON to application data [306](#)  
Transforming JSON using the linkable interface [197](#)  
transforming XML [384](#)  
transforming XML to binary data [392](#)  
troubleshooting [451](#)  
trust client  
    invoking [435](#)

## U

URI  
    for IBM MQ transport [39](#)  
utility program  
    JSON assistant [197](#)  
    web services assistant [140](#)  
    XML assistant [311](#)

## V

validating SOAP messages [191](#)  
Variable arrays [296](#), [368](#)

## W

web service errors [442](#)  
Web Services Addressing  
    <wsa:Action> [105](#)  
    addressing handler [101](#), [102](#)  
    default actions [103](#), [106](#)  
    default EPR [103](#), [105](#)  
    DFHWS-URI [98](#)  
    DFHWSADH [101](#), [102](#)  
    EPR [98](#)  
    explicit actions [103](#), [105](#)  
    MAP [98](#)  
    parameters [170](#)  
    provider pipeline configuration [102](#)  
    requester pipeline configuration [101](#)  
    requester service [103](#)  
    WSADDR-EPR-ANY [170](#)  
    WSDL 1.1 [106](#)  
web services assistant  
    creating a service provider application [129](#), [173](#)  
web services discovery [36](#)  
WS-Addressing  
    <wsa:Action> [105](#)  
    addressing handler [101](#), [102](#)  
    default actions [106](#)  
    default EPR [105](#)  
    DFHWS-URI [98](#)  
    DFHWSADH [101](#), [102](#)  
    EPR [98](#)  
    explicit actions [105](#)  
    MAP [98](#)  
    parameters [170](#)  
    provider pipeline configuration [102](#)  
    requester pipeline configuration [101](#)  
    WSADDR-EPR-ANY [170](#)  
    WSDL 1.1 [106](#)  
WSDL  
    and application data structure [14](#)  
    converting to language structure [154](#), [210](#), [222](#), [320](#)

WSDL (*continued*)  
  querying [36](#)  
  Web Services Addressing [103](#)  
WSDL 1.1  
  default EPR [105](#)  
WSDL document  
  variable-length [372](#)  
  white space [372](#)

## X

XML  
  assistant [311](#)  
  data types [384](#)  
  parsing [384](#)  
  querying [383](#)  
  transform to data [311](#)  
  transforming [384](#)  
  xsd:any data types [385](#)  
XML assistant  
  DFHLS2SC [388](#)  
XML binding [388](#)  
XML schema  
  variable-length [372](#)  
  white space [372](#)  
XML Schema [337](#), [349](#), [353](#), [358](#), [362](#)  
xsd:any data types [385](#)





