CICS Transaction Server for z/OS

IBM

Release Guide

Version 3 Release 1

CICS Transaction Server for z/OS

Release Guide

Version 3 Release 1

Contents

Preface

What this book is about

This book provides information about new and changed function in CICS®
Transaction Server for z/OS®, Version 3 Release 1. It gives an overview of the
changes to reference information, and points you to the manuals where more
detailed reference information is given.

The programming interface information given in this book is intended to show only
what is new and changed from the previous release of CICS TS, and to highlight
the benefits of the new function. For programming interface information, read the
primary sources of programming interface and associated information in the
following publications:

- *CICS Application Programming Reference*
- *CICS System Programming Reference*
- *CICS Customization Guide*
- *CICS External Interfaces Guide*
- *CICSPlex SM Application Programming Guide*
- *CICSPlex SM Application Programming Reference*

Who this book is for

This book is for those responsible for the following user tasks:

- Evaluation and planning
- System administration
- Programming
- Customization

What you need to know to understand this book

The book assumes that you are familiar with CICS and CICSPlex SM, either as a
systems administrator, or as a systems or application programmer.

Notes on terminology

When the term "CICS" is used without any qualification in this book, it refers to the
CICS element of IBM CICS TS.

"CICSPlex SM" is used for the CICSPlex System Manager element of IBM CICS
TS.

"MVS" is used for the operating system, which is a base element of z/OS.

xi

Syntax notation

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

The conventions used in the syntax notation are:

Notation	Explanation
►►─┬─A─┬─────────────►◄ 　　├─B─┤ 　　└─C─┘	Denotes a set of required alternatives. You must specify one (and only one) of the values shown.
►►─┬─A─┬─────────────►◄ 　　├─B─┤ 　　└─C─┘	Denotes a set of required alternatives. You must specify at least one of the values shown. You can specify more than one of them, in any sequence.
►►─┬───┬─────────────►◄ 　　├─A─┤ 　　├─B─┤ 　　└─C─┘	Denotes a set of optional alternatives. You can specify none, or one, of the values shown.
►►─┬───┬─────────────►◄ 　　├─A─┤ 　　├─B─┤ 　　└─C─┘	Denotes a set of optional alternatives. You can specify none, one, or more than one of the values shown, in any sequence.
►►─┬─A─┬─────────────►◄ 　　├─B─┤ 　　└─C─┘	Denotes a set of optional alternatives. You can specify none, or one, of the values shown. **A** is the default value that is used if you do not spacify anything.
►►─┤ Name ├──────────►◄ **Name:** ├─A─┬───────────────┤ 　　└─B─┘	A reference to a named section of syntax notation.
►►──A=*value*─────────►◄	**A**= denote characters that should be entered exactly as shown. *value* denotes a variable, for which you should specify an appropriate value.
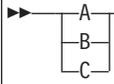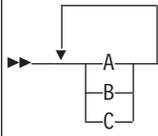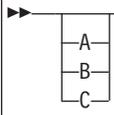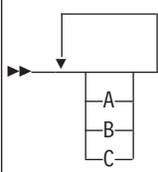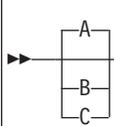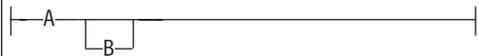
Part 1. Summary of CICS Transaction Server for z/OS, Version 3 Release 1

This part contains a brief overview of the major new function in CICS Transaction Server for z/OS, Version 3 Release 1.

1

Chapter 1. Summary of CICS Transaction Server for z/OS, Version 3 Release 1

The major new and changed function in CICS Transaction Server for z/OS, Version 3 Release 1 are represented in the three themes of *CICS Integration*, *Application transformation*, and *Enterprise management*. CICS TS 3.1 also includes miscellaneous changes that are outside the scope of the three themes, and a number of functions are discontinued or reduced in scope.

CICS integration

CICS TS 3.1 provides a range of new and enhanced capabilities which enable reuse of existing CICS applications within broader on-demand scenarios, by the use of widely adopted application programming interfaces and standard protocols. These include:

Support for Web services

> Support for Web services is fully integrated into CICS TS 3.1. In this release, CICS applications can act in the role of both service provider and service requester, where the services are defined using Web Services Description Language (WSDL).
>
> The infrastructure provided as part of CICS TS V3.1 includes a distributed transaction coordination capability compatible with the WS-AtomicTransaction specification.
>
> The support for Web services includes the *CICS Web services assistant*, a batch utility which can help you to
>
> * transform an existing CICS application into a Web service
> * and enable a CICS application to use a Web service provided by an external provider.
>
> 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*.
>
> For more information, see Chapter 2, "Web services in CICS," on page 11.

Support for HTTP client requests from CICS applications
> In CICS TS 3.1 the ability of CICS to act as an HTTP client is fully integrated into CICS Web support. Your application programs can now use **EXEC CICS** commands to open a connection to a server, to make an HTTP request, and to receive the response.
>
> For more information, see Chapter 3, "Support for HTTP client requests from CICS applications," on page 83.

Support for HTTP/1.1
> CICS Web support is now conditionally compliant with the HTTP/1.1 specification, as defined by RFC 2616, *Hypertext Transfer Protocol -- HTTP/1.1*.
>
> New functions that are supported in CICS TS 3.1 include *chunked transfer-coding*, *pipelining*, and *persistent connections*.

3

For more information, see Chapter 4, "CICS Web support upgrade to HTTP/1.1," on page 111.

Other enhancements to CICS Web support

Other enhancements to CICS Web support in CICS TS 3.1 include:

* The URIMAP resource, which gives improved capability for processing HTTP requests and responses when CICS is an HTTP server
* Improvements to the way CICS Web supports code pages (including support of the UTF-8 and UTF-16 code pages)
* A new API command which provides support for HTTP time and date formats.

For more information, see Chapter 5, "General enhancements to CICS Web support," on page 131.

Improvements to support for SSL

Support for the Secure Sockets Layer (SSL) is enhanced in CICS TS 3.1:

* CICS now supports the Transport Layer Security (TLS) 1.0 protocol as well as SSL 3.0, allowing you to use the new AES cipher suites that offer 128-bit and 256-bit encryption.
* The number of simultaneous SSL connections that can be used in the system at one time has increased to achieve better throughput.
* There is more flexibility in controlling the encryption negotiation between client and server. You can now specify a minimum as well as a maximum encryption level in CICS for negotiating with particular users.
* CICS can now check all certificates against a certificate revocation list when negotiating with clients. Any connections using revoked certificates are closed immediately.
* You can specify whether you want to share session IDs across a sysplex by using the SSL cache. CICS will perform a partial SSL handshake if the client has negotiated with CICS previously. If the cache is shared across a number of CICS regions, this will improve the performance of SSL negotiation and connection throughput.

For more information, see Chapter 6, "Improvements to Internet security," on page 179.

Application transformation

CICS Transaction Server for z/OS, Version 3 Release 1 provides a range of new functions that enable you to enhance your existing applications, and to construct new applications, using contemporary programming languages, constructs, and tools. These include:

Enhanced inter-program data transfer: channels as modern-day COMMAREAs

In CICS TS 3.1, *channels* and *containers* provide an improved method of transferring data between programs, in amounts that far exceed the 32KB limit that applies to COMMAREAs.

* A *container* is a named block of data designed for passing information between programs.
* Containers are grouped together in sets called *channels*. A channel is a standard mechanism for exchanging data between CICS programs, and is analogous to a parameter list. A channel can be used on the LINK, START, XCTL, and RETURN commands, and with local and remote

transactions. There is no limit to the number of containers that can be added to a channel, and the size of each container is limited only by the amount of storage available.

The channel and container model has several advantages over the traditional COMMAREA model. These include:

- There is no 32KB size limit.
- Unlike a COMMAREA, which is a monolithic block of data, a channel with several containers can represent structured data.

For more information, see Chapter 7, "Enhanced inter-program data transfer: channels as modern-day COMMAREAs," on page 197.

OPENAPI support

CICS extends the use of Open Transaction Environment (OTE) functionality by providing support for OPENAPI application programs. Prior to this, OPENAPI function was available only to task related user exits (TRUEs). The use of OPENAPI programs allows application workloads to be moved off the QR TCB onto multiple open TCBs.

For more information, see Chapter 8, "OPENAPI Support," on page 255.

XPLINK support for C and C++

New support for C and C++ has been introduced in CICS TS 3.1, bringing the performance of applications that are written in these languages, to a level comparable with that obtained with COBOL, PL/I, or Assembler. The performance improvements are provided by the Extra Performance Linkage (XPLINK) feature of z/OS, which provides high performance subroutine linkage mechanisms and guard pages for stack extension, resulting in highly optimized execution path lengths.

The applications that benefit most from these improvements are those that:

- are run in the CICS Open Transaction Environment (OTE)
- are written to threadsafe standards and use only threadsafe CICS commands

The use of XPLINK enables the latest compiler and optimization technologies included with C and C++ to be exploited. In particular, you can achieve greater reuse of C and C++ code by using dynamic load libraries (DLLs) that were compiled with the XPLINK option in the CICS environment.

For more information, see Chapter 9, "XPLink Support," on page 261.

Support for Language Environment® enabled Assembler Main programs

CICS TS 3.1 supports main programs which are written in Assembler, and that are Language Environment enabled. This support extends the availability of Language Environment use, and makes Debugger support available with Assembler programs.

For more information, see Chapter 10, "Support for Language Environment conforming assembler MAIN programs," on page 271.

Enterprise management

The CICSPlex® SM element of CICS TS 3.1 provides new capabilities that enable effective management of large runtime configurations by the use of modern user interfaces, so that you can meet your demanding service level objectives. These include:

Enhancements to the CICSPlex SM Web User Interface

The CICSPlex SM Web User Interface has been improved to make it more powerful and more usable. The Web User Interface is now functionally equivalent to the CICSPlex SM TSO end user interface, and is now the primary method of accessing CICSPlex SM.

For more information, see Chapter 11, "CICSPlex SM Web User Interface enhancements," on page 285.

Enhancements to the CICSPlex SM batched repository update facility

New facilities have been introduced that provide alternatives to the existing TSO EUI command for submitting batched updates to a specified CICSPlex SM repository. The alternative BATCHREP facilities introduced are:

- A new BATCHREP resource table that is available as an object for reference by the CPSM API
- Web User Interface support of the new resource table
- A batch utility program that provides a BATCHREP facility

For more information, see Chapter 12, "Enhancements to CICSPlex SM batched repository update facility," on page 307.

Miscellaneous changes

Other new and changed functions in CICS Transaction Server for z/OS, Version 3 Release 1 include:

EXTRACT STATISTICS command

A new SPI command, `EXTRACT STATISTICS` performs a function equivalent to `COLLECT STATISTICS`, for the URIMAP, PIPELINE, and WEBSERVICE resources.

For more information, see Chapter 14, "EXTRACT STATISTICS command," on page 315.

Support for mixed case passwords

When the security manager used with CICS supports the use of mixed case passwords, CICS TS 3.1 does not convert passwords to uppercase before passing them to the security manager.

For more information, see Chapter 15, "Support for mixed case passwords," on page 319.

Simplified definition of default code pages

In CICS TS 3.1, the default client or server code pages used by the DFHCNV data conversion table can be defined in the system initialization parameters.

For more information, see Chapter 17, "Simplified definition of default code pages," on page 327.

64-Bit Addressing toleration

CICS does not support 64-bit addressing execution, but programs can use storage at addresses which are only available when CICS is running on 64-bit architecture machines. The CICS abend capture mechanisms have changed so that the contents of the full 64-bit general purpose registers is captured.

For more information, see Chapter 18, "64-Bit Addressing Toleration changes," on page 329.

Support for revoked user IDs

When the `EXEC CICS VERIFY PASSWORD` command is issued, CICS now honors the revoked status of a user ID or a user's group connection.

Discontinued function

Some functions which were supported in CICS Transaction Server for z/OS, Version 2 have been discontinued, or reduced in scope in CICS TS 3.1. They include:

Runtime support for OS/VS COBOL programs

OS/VS COBOL programs, which had runtime support in CICS Transaction Server for z/OS, Version 2, cannot run under CICS TS 3.1.

For more information, see Chapter 20, "Withdrawal of runtime support for OS/VS COBOL programs," on page 335.

Support for BTAM and TCAM

CICS support for the Basic Telecommunications Access Method (BTAM) is discontinued in CICS TS 3.1. Support for the Telecommunications Access Method (TCAM) is limited to indirect support for the DCB interface.

For more information, see Chapter 21, "Changes to BTAM and TCAM support," on page 337.

Support for 1-byte console IDs

Support for the 1-byte console id has been removed. You can define consoles using the CONSNAME(*name*) attribute on the TERMINAL definition.

For more information, see Chapter 22, "Withdrawal of support for 1-byte console id," on page 341.

The CICS connector for CICS TS

Support for the CICS Connector for CICS TS, introduced in CICS Transaction Server for z/OS, Version 2 Release 1, is withdrawn.

For more information, see Chapter 23, "Withdrawal of the CICS Connector for CICS TS," on page 343.

Support for Java™ program objects and hot-pooling

Run-time support for Java program objects and for hot-pooling (HPJ) is withdrawn.

For more information, see Chapter 24, "Withdrawal of run-time support for Java program objects and hot-pooling," on page 345.

CICSPlex SM support for Windows remote MAS

For more information, see Chapter 25, "Removal of CICSPlex SM support for Windows remote MAS," on page 349.

Withdrawal of the CICS Transaction Affinities Utility

The CICS Transaction Affinities Utility is no longer supplied with CICS. Its function of detecting transaction affinities is now provided by the CICS Interdependency Analyzer, which is a more sophisticated tool.

For more information, see Chapter 26, "Withdrawal of the CICS Transaction Affinities Utility," on page 351.

The CICS Information Center

In CICS TS 3.1, the CICS Information Center runs within the WebSphere® Studio WorkBench User Assistance system, an Eclipse framework that contains a number of documentation plug-ins that make up the information center. The new look and feel of the information center, in particular the welcome page, is now consistent with other product information centers.

For more information, see Chapter 30, "The Eclipse information center," on page 371

Part 2. CICS integration

CICS Transaction Server for z/OS, Version 3 Release 1 provides a range of new and enhanced capabilities which enable reuse of existing CICS applications within broader on demand scenarios, by the use of widely adopted application programming interfaces and standard protocols.

Chapter 2. Web services in CICS

Support for Web services is fully integrated into CICS Transaction Server for z/OS, Version 3 Release 1. In this release, CICS applications can act in the role of both service provider and service requester, where the services are defined using Web Services Description Language (WSDL).

The infrastructure provided as part of CICS TS V3.1 includes a distributed transaction coordination capability compatible with the WS-AtomicTransaction
\# specification, and support for the Web Services Security: SOAP Message Security
\# 1.0 specification.

\# **Note:** Support for Web Services Security is delivered in the *CICS WS-Security*
\# *Component*. To install the component, apply the PTF for APAR PK22736.

\# Additionally, the IBM® XML Toolkit for z/OS V1.9 is required for WS-Security
\# support. For information about this no-charge product, see "Software
\# Requirements" on page 355.

The support for Web services includes the *CICS Web services assistant*, a batch utility which can help you to
- transform an existing CICS application into a Web service
- and enable a CICS application to use a Web service provided by an external provider.

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*.

How Web services can help your business

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

Web services can help your business by:
- Reducing the cost of doing business
- Making it possible to deploy solutions more rapidly
- Opening up new opportunities.

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

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

Web services terminology

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

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

Service provider
The collection of software that provides a Web service.

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

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

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

Simple Object Access Protocol
See SOAP.

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

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

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

The specification for SOAP 1.1 is published at http://www.w3.org/TR/SOAP.

The specification for SOAP 1.2 is published at:

http://www.w3.org/TR/soap12-part0
http://www.w3.org/TR/soap12-part1
http://www.w3.org/TR/soap12-part2

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

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

SOAP node
Processing logic which operates on a SOAP message.

SOAP receiver
A SOAP node that accepts a SOAP message.

SOAP sender
A SOAP node that transmits a SOAP message.

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

UDDI Universal Description, Discovery and Integration

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

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

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

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

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

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

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

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

Web Services Security
A set of enhancements to SOAP messaging that provides message integrity and confidentiality. The specification is published by OASIS

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

WS-Atomic Transaction
Web Services Atomic Transaction

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

WSDL Web Service Description Language.

WSS Web Services Security

XML Extensible Markup Language.

The specifications for XML are published at:

> http://www.w3.org/TR/soap12-part0
>
> http://www.w3.org/TR/soap12-part1
>
> http://www.w3.org/TR/soap12-part2

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

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

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

Requirements

There are no special hardware or software requirements to support this function.

Related information:

Chapter 27, "The CICS operating environment," on page 355

How CICS supports Web services

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

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

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

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

Web service, you can start with its Web service description and generate a high level language structure that you can use in your program.

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

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

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

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

Message handlers and pipelines

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

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

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

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

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

Figure 1 shows a pipeline of three message handlers:
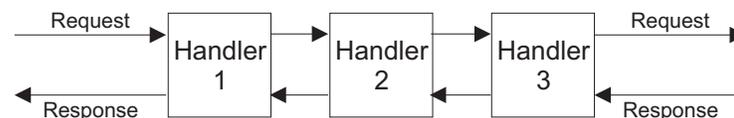
Figure 1. A generic CICS pipeline

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

In the request phase
1. Handler 1
2. Handler 2
3. Handler 3

In the response phase
1. Handler 3
2. Handler 2
3. Handler 1

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

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

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

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

Transport-related handlers

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

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

This is illustrated in Figure 2:
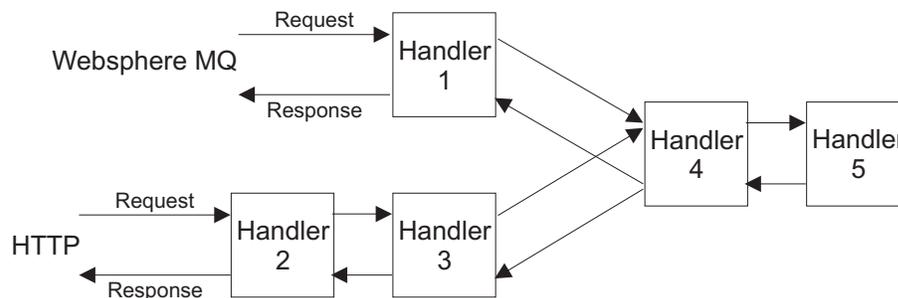
Figure 2. Pipeline with transport-related handlers

In this example, which applies to a service provider:

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

• Handler 5 is the terminal handler.

Interrupting the flow

During processing of a request, a message handler can decide not to pass a message to the next handler, but can, instead, generate a response. Normal processing of the message is interrupted, and some handlers in the pipeline are not invoked. For example, suppose that handler 2 in Figure 3 is responsible for performing security checks.
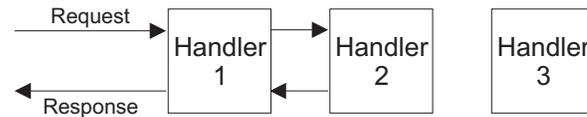
Figure 3. Interrupting the pipeline flow

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

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

A service provider pipeline

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

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

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

Figure 4 on page 18 illustrates a pipeline of three message handlers in a service provider setting:
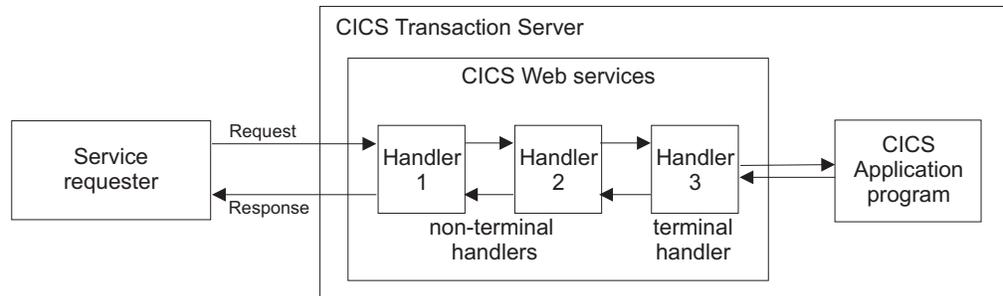
Figure 4. A service provider pipeline

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 19 illustrates a pipeline of three message handlers in a service requester setting:
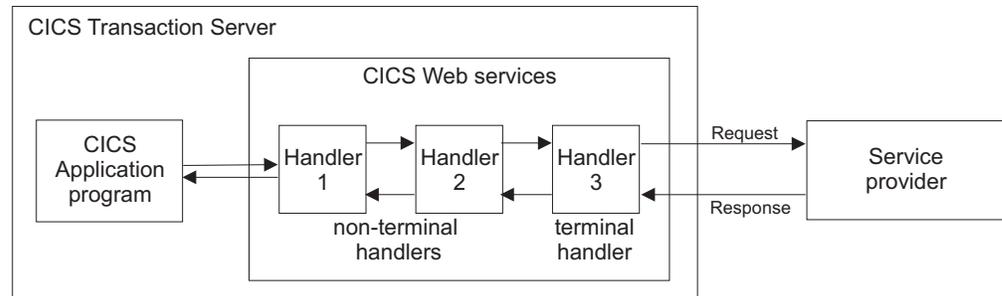
Figure 5. A service requester pipeline

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.

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

  You cannot configure a CICS pipeline to function as a SOAP intermediary.
• You can configure a CICS pipeline to have more than one SOAP message handler.
• The CICS-provided SOAP message handlers can be configured to invoke one or more user-written header-handling routines.

- The CICS-provided SOAP message handlers can be configured to enforce some aspects of compliance with the WS-I Basic Profile 1.1 and Simple SOAP Binding Profile 1.0, and to enforce the presence of particular headers in the SOAP message.

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

SOAP messages and the application data structure

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

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

- The Web service binding file

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

- The Web service description

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

Figure 6 shows where these files are used in a service provider.
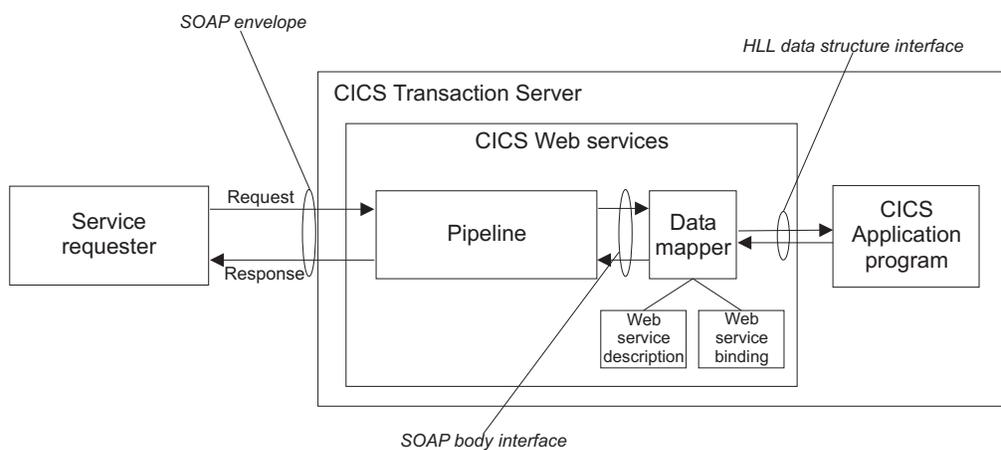
Figure 6. Mapping the SOAP body to the application data structure in a service provider

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 7 shows where these files are used in a service requester.
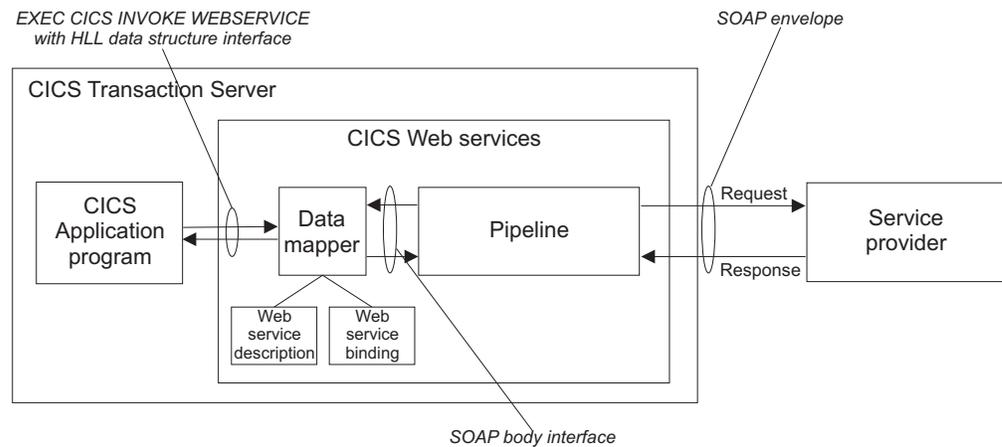
Figure 7. Mapping the SOAP body to the application data structure in a service requester

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

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

There are some situations in which, even though you are using SOAP messages, you cannot use the transformation that the CICS Web services assistant generates:
- When the same data cannot be represented in the SOAP message and in the high level language.

  All the high level languages that CICS supports, and XML Schema, support a variety of different data types. However, there is not a one-to-one correspondence between the data types used in the high level languages, and those used in XML Schema, and there are cases where data can be represented in one, but not in the other. In this situations, you should consider one of the following:
  – Change your application data structure. This may not be feasible, as it might entail changes to the application program itself.
  – Construct a wrapper program, which transforms the application data into a form that CICS can then transform into a SOAP message body. If you do this, you can leave your application program unchanged. In this case CICS Web service support interacts directly with the wrapper program, and only indirectly with the application program.
- When your application program is in a language which is not supported by the CICS Web services assistant.

  In this situation, you should consider one of the following:
  – Construct a wrapper program that is written in one of the languages that the CICS Web services assistant does support (COBOL, PL/I, C or C++).

– Instead of using the CICS Web services assistant, write your own program to perform the mapping between the SOAP messages and the application program's data structure.

WSDL and the application data structure

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

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

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

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

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

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

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

The CICS Web services assistant consists of two utility programs:

DFHWS2LS
> This utility program takes a Web service description as a starting point. It uses the descriptions of the messages, and the data types used in those messages, to construct high level language data structures that you can use in your application programs.

DFHLS2WS
> This utility program takes a high level language data structure as a starting point. It uses the structure to construct a Web services description that contains descriptions of messages, and the data types used in those messages derived from the language structure.

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

An example of COBOL to WSDL mapping

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

Figure 8 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 8. 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 9:

```xml
<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 9. WSDL fragment derived from a COBOL data structure

The Web service binding file

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

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

Web service binding files are created:
- By utility program DFHWS2LS when language structures are generated from WSDL.
- By utility program DFHLS2WS when WSDL is generated from a language structure.

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

External standards

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

Extensible Markup Language Version 1.0

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

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

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

SOAP 1.1 and 1.2

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

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

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

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

SOAP 1.2 is a W3C recommendation and is published in two parts:
- Part 1: Messaging Framework is published at http://www.w3.org/TR/soap12-part1/ .
- Part 2: Adjuncts is published at http://www.w3.org/TR/soap12-part2/.

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

Web Services Description Language Version 1.1

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

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

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

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

Web Services Coordination Version 1.0

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

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

The specification for WS-Coordination is published at http://www.ibm.com/developerworks/library/specification/ws-tx/.

Web Services Atomic Transaction Version 1.0

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

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

The specification for WS-AtomicTransaction is published at http://www.ibm.com/
developerworks/library/specification/ws-tx/.

WS-I Basic Profile Version 1.1

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

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

To summarize, the WS-I Basic Profile Version 1.0 has now been split into two
separately published profiles. These are:
* WS-I Basic Profile Version 1.1
* WS-I Simple SOAP Binding Profile Version 1.0

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

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

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

WS-I Simple SOAP Binding Profile Version 1.0

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

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

WS-I Basic Profile 1.0 has now been split into two separately published profiles.
These are:
* WS-I Basic Profile Version 1.1
* WS-I Simple SOAP Binding Profile Version 1.0

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

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

Web Services Security: SOAP Message Security

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

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

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

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

The specification is published by the Organization for the Advancement of Structured Information Standards (OASIS) at http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

Web Services Security: UsernameToken Profile 1.0

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

The WSS: UsernameToken Profile 1.0 specification is published by the Organization for the Advancement of Structured Information Standards (OASIS) at http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf.

Web Services Security: X.509 Certificate Token Profile 1.0

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

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

XML Encryption Syntax and Processing

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

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

XML-Signature Syntax and Processing

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

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

The specification for XML-Signature is published by World Wide Web Consortium (W3C) at http://www.w3.org/TR/xmldsig-core.

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

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

How CICS complies with WS-I Basic Profile 1.1:

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

See High level language and XML schema mapping for a list of unsupported schema elements.

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

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

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

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

INSTANCE
Software that implements a `wsdl:port`.

CONSUMER
Software that invokes an INSTANCE.

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

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

How CICS complies with Web Services Security specifications:

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

Compliance with Web Services Security: SOAP Message Security

Security header

The `<wsse:Security>` header provides a mechanism for attaching security-related information targeted at a specific recipient in the form of a SOAP actor or role. This could be the ultimate recipient of the message or an intermediary. The following attributes are supported in CICS:

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

Security tokens

The following security tokens are supported in the security header:

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

Token references

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

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

Signature algorithms

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

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

Signature signed parts

CICS allows the following SOAP elements to be signed:

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

Encryption algorithms

The following data encryption algorithms are supported:

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

The following key encryption algorithm is supported:

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

Encryption message parts
> CICS allow the following SOAP elements to be encrypted:
> * the SOAP body

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

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

Compliance with Web Services Security: UsernameToken Profile 1.0

The following aspects of this specification are supported:

Password types
> Text

Token references
> Direct reference

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

The following aspects of this specification are supported:

Token types
> * X.509 Version 3: Single certificate. See http://docs.oasis-open.org/wss/ 2004/01/oasis-200401-wss-x509- token-profile-1.0#X509v3.
> * X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL). See http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss- x509- token-profile-1.0#X509PKIPathv1.

- X.509 Version 3: PKCS7 with or without CRLs. The IBM Software Development Kit (SDK) supports both. The Sun Java Development Kit (JDK) supports PKCS7 without CRL only.

Token references

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

Aspects that are not supported

The following items are not supported in CICS:

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

Planning to use Web services

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

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

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

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

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

Do you intend to use SOAP messages?

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

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

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

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

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

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

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

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

Next steps:
- Planning a service provider
- Planning a service requester

Planning a service provider application

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

Figure 10 on page 33 shows a typical application which is partitioned to ensure a separation between communication logic and business logic.
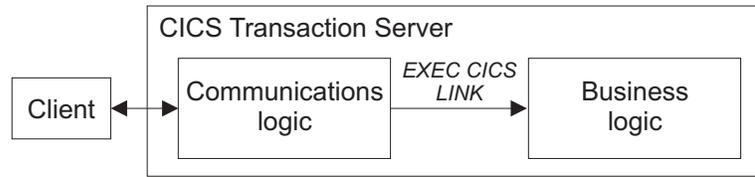
Figure 10. Application partitioned into communications and business logic

In many cases, you can deploy the business logic directly as a service provider application. This is illustrated in Figure 11.
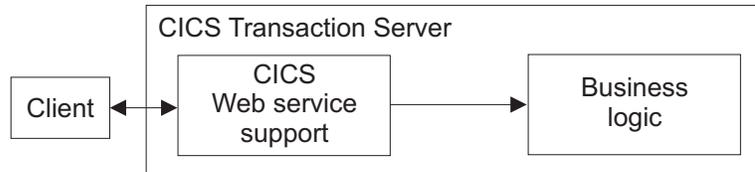
Figure 11. Simple deployment of CICS application as a Web service provider

To use this simple model, the following conditions apply:

When you are using the CICS Web services assistant to generate the mapping between SOAP messages and application data structures:
> The data types used in the interface to the business logic must be supported by the CICS Web services assistant. If this is not the case, you must interpose a wrapper program between CICS Web service support and your business logic.

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

When you are not using the CICS Web services assistant:
> Message handlers in your service provider pipeline must interact directly with your business logic.

Using a wrapper program

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

Figure 12. Deployment of CICS application as a Web service provider using a wrapper program

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

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

Error handling

If you are planning to use the CICS Web services assistant, you should also consider how to handle rolling back changes when errors occur. When a SOAP request message is received from a service requester, the SOAP message is transformed by CICS just before it is passed to your application program. If an error occurs during this transformation, CICS does not automatically roll back any work that has been performed on the message. For example, if you plan to add some additional processing on the SOAP message using handlers in the pipeline, you need to decide if they should roll back any recoverable changes that they have already performed.

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

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

Planning a service requester application

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

Figure 13 shows a typical application which is partitioned to ensure a separation between communication logic and business logic. The application is ideally structured for reuse of the business logic in a Web service requester.

Figure 13. Application partitioned into communications and business logic

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

- When you are using the CICS Web services assistant to generate the mapping between SOAP messages and application data structures, you must use an **EXEC CICS INVOKE WEBSERVICE** command, and pass the application's data structure to

CICS Web services support. Also, the data types used in the interface to the business logic must be supported by the CICS Web services assistant.

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

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

Either way, it follows that your business logic cannot invoke a Web service directly unless you are prepared to change the program. For the Web services assistant, this option is shown in Figure 14, but it is not advisable in either case.
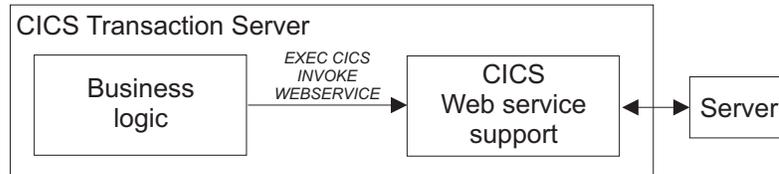
Figure 14. Simple deployment of CICS application as a Web service requester

Using a wrapper program

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

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

For the case when the Web services assistant is used, this structure is illustrated in Figure 15.
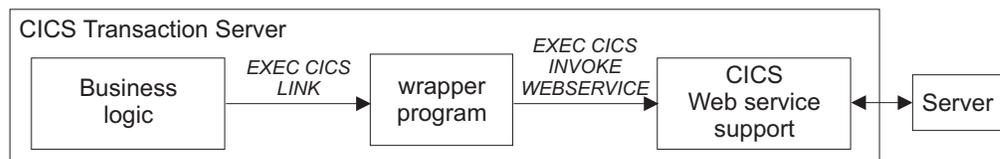
Figure 15. Deployment of CICS application as a Web service requester using a wrapper program

Error handling

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

If you are planning to implement Web service atomic transactions in your requester application program, the error handling is different. If the remote service provider encounters an error and rolls back its changes, a SOAP fault message is returned

and the local transaction in CICS also rolls back. If local optimization is in effect, the
service requester and provider use the same transaction. If the provider encounters
an error, any changes made by the transaction in the requester are also rolled
back.

The CICS Web services assistant

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

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

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

basicsoap11provider.xml
> Pipeline configuration file for a service provider using the SOAP 1.1 message handler.

basicsoap11requester.xml
> Pipeline configuration file for a service requester using the SOAP 1.1 message handler.

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

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

You don't want to use SOAP messages
> If you prefer to use a non-SOAP protocol for your messages, you can do so. However, your application programs will be responsible for parsing inbound messages, and constructing outbound messages.

You want to use SOAP messages, but don't want CICS to parse them
> For an inbound message, the assistant maps the SOAP body to an application data structure. In some applications, you may want to parse the SOAP body yourself.

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

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

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

The CICS Web services assistant comprises two utility programs:

DFHLS2WS
> Generates a Web service binding file from a language structure. This utility also generates a Web service description.

DFHWS2LS
> Generates a Web service binding file from a Web service description. This utility also generates a language structure that you can use in your application programs.

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

DFHLS2WS: high level language to WSDL conversion

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

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

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

Job control statements for DFHLS2WS

JOB   Initiates the job.

EXEC   Specifies the procedure name (DFHLS2WS).

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

INPUT.SYSUT1 DD
> Specifies the input. The input parameters are usually specified in the input stream. However, they can be defined in a data set, or in a member of a partitioned data set.

Symbolic parameters

The following symbolic parameters are defined in cataloged procedure DFHLS2WS:

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

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

PATHPREF=*prefix*

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

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

SERVICE=*value*

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

TMPDIR=*tmpdir*

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

The default value is /tmp.

TMPFILE=*tmpprefix*

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

The default value is LS2WS

USSDIR=*path*

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

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

The temporary work space

DFHLS2WS creates the following three temporary files during execution:

> *tmpdir*/*tmpprefix*.in
>
> *tmpdir*/*tmpprefix*.out
>
> *tmpdir*/*tmpprefix*.err

where

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

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

> /tmp/LS2WS.in
>
> /tmp/LS2WS.out
>
> /tmp/LS2WS.err

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

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

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

Input parameters for DFHLS2WS

If you need any help understanding this syntax diagram, see "Syntax notation" on page xii.

```
►►──PDSLIB=value───────────────────────────────────────────────────────────────►
              └─PDSCP=value─┘ └─REQMEM=value─┘ └─RESPMEM=value─┘

#  ►──┬─LANG=COBOL──────────┬─────────────────────────────────────────────────►
       ├─LANG=PLI-ENTERPRISE─┤
       ├─LANG=PLI-OTHER──────┤
       ├─LANG=C──────────────┤
       └─LANG=CPP────────────┘ └─STRUCTURE=(─┬─DFHREQUEST─┬─,─┬─DFHRESPONSE─┬─)─┘
                                             └─request────┘   └─response────┘

                                                        ┌─PGMINT=CHANNEL──────────┐
#                                                       │         └─CONTID=value─┘ │
   ►─PGMNAME=value──────────────────────────────────────┴──────────────────────────►
              └─TRANSACTION=name─┘ └─USERID=id─┘ └─URI=value─┘ └─PGMINT=COMMAREA──┘

#                                         ┌─MAPPING-LEVEL=1.0──────────────────────┐
   ►─WSBIND=value──WSDL=value──LOGFILE=value─┼─MAPPING-LEVEL=1.1──────────────────────┼─►
                                           └─MAPPING-LEVEL=1.2─┬─CHAR-VARYING=NO───┬─┘
                                                               └─CHAR-VARYING=NULL─┘

#     ┌─MINIMUM-RUNTIME-LEVEL=MINIMUM──┐
   ►──┼─MINIMUM-RUNTIME-LEVEL=1.0──────┼────────────────────────────────────────────►
       ├─MINIMUM-RUNTIME-LEVEL=1.1──────┤ └─CCSID=value─┘ └─REQUEST-NAMESPACE=value─┘ └─RESPONSE-NAMESPACE=value─┘
       ├─MINIMUM-RUNTIME-LEVEL=1.2──────┤
       └─MINIMUM-RUNTIME-LEVEL=CURRENT──┘

#  ►──┬───────────────────┬──┬─────────────────┬──────────────────────────────────►◄
       ├─SYNCONRETURN=─┬─NO─┤  ├─WSDLCP=LOCAL─┤
       └─SYNCONRETURN=YES───┘  └─WSDLCP=UTF-8─┘
```

Parameter use

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

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

is equivalent to

```
WSBIND=wsbinddir/app1
```

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

Parameter descriptions

CCSID=*value*
> Specifies the CCSID that is used at run time to encode character data in the application data structure. The value of this parameter overrides the value of the **LOCALCCSID** system initialization parameter. The *value* must be an EBCDIC CCSID that is supported by Java and z/OS conversion services. If you do not specify this parameter, the application data structure is encoded using the CCSID specified in the system initialization parameter.
>
> You can use this parameter with any mapping level. However, if you want to deploy the generated files, you must apply APAR PK23547 to the CICS region to achieve the minimum runtime level of code to install the Web service binding file.

CHAR-VARYING=NO|NULL
> Specifies how character fields in the language structure should be mapped when the mapping level is 1.2. A character field in COBOL is a Picture clause of type X, for example PIC(X) 10; a character field in C/C++ is a character array. This parameter does not apply to Enterprise and Other PL/I language structures. The options you can select are:
>
> **NO** Character fields are mapped to an xsd:string and are processed as fixed length fields. The maximum length of the data is equal to the length of the field.
>
> **NULL** Character fields are mapped to an xsd:string and are processed as null terminated strings. CICS adds a terminating null character when transforming from a SOAP message. The maximum length of the character string is calculated as one character less than the length indicated in the language structure.

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

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

LANG=PLI-ENTERPRISE
> Specifies that the programming language of the high level language structure is Enterprise PL/I.

LANG=PLI-OTHER
> Specifies that the programming language of the high level language structure is a level of PL/I other than Enterprise PL/I.

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

LANG=**CPP**
>   Specifies that the programming language of the high level language structure is
>   C++.

LOGFILE=*value*
>   The fully qualified HFS name of the file into which DFHLS2WS writes its activity
>   log and trace information. DFHLS2WS creates the file (but not the directory
>   structure) if it does not already exist.
>
>   Normally, you will not need to use this file, but it may be requested by the IBM
>   service organization if you encounter problems with DFHLS2WS.

\# **MAPPING-LEVEL**=**{1.0|1.1|1.2}**
>   Specifies the level of mapping that DFHLS2WS should use when generating the
>   Web service binding file and Web service description. This parameter is
>   available when you apply APAR PK15904. You also need to apply APAR
>   PK23547 if you want to use the 1.2 mapping level option. The options you can
>   select are:
>
>   **1.0**   This is the default mapping level.
>
>   **1.1**   Use this mapping if you need to regenerate a binding file at this specific
>   level.
>
>   **1.2**   At this mapping level you can use the parameter **CHAR-VARYING** to
>   control how character arrays should be processed at run time.
>   VARYING and VARYINGZ arrays are also supported in PL/I.
>
>   For details of what is supported at each level of mapping, see Mapping levels
>   for the CICS Web services assistant.

\# **MINIMUM-RUNTIME-LEVEL**=**{MINIMUM|1.0|1.1|1.2|CURRENT}**
>   Specifies the minimum CICS runtime environment that the Web service binding
>   file can be deployed into. If you select a level that does not match the other
>   parameters that you have specified, you receive an error message. The options
>   you can select are:
>
>   **MINIMUM**
>   The lowest possible runtime level of CICS is allocated automatically
>   given the parameters that you have specified.
>
>   **1.0**   The generated Web service binding file deploys successfully into a
>   CICS TS 3.1 region that does not have APARs PK15904 and PK23547
>   applied. You cannot specify the **CHAR-VARYING**, **CCSID**, or **MAPPING-LEVEL**
>   parameters.
>
>   **1.1**   The generated Web service binding file deploys successfully into a
>   CICS TS 3.1 region that has at least APAR PK15904 applied. You
>   cannot specify the **CHAR-VARYING** or **CCSID** parameters. You cannot use a
>   mapping level of 1.2 for the **MAPPING-LEVEL** parameter.
>
>   **1.2**   The generated Web service binding file deploys successfully into a
>   CICS TS 3.1 region that has both APAR PK15904 and PK23547
>   applied. You can use any optional parameter at this level.
>
>   **CURRENT**
>   The generated Web service binding file deploys successfully into a
>   CICS region at the same runtime level as the one you are using to
>   generate the Web service binding file.

PDSLIB=*value*
>   Specifies the name of the partitioned data set that contains the high level

language data structures to be processed. The data set members used for the request and response are specified in the `REQMEM` and `RESPMEM` parameters respectively.

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

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

`PGMINT=CHANNEL|COMMAREA`
For a service provider, specifies how CICS passes data to the target application program:

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

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

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

\# 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 target CICS application program that will be exposed as a Web service. This is the program that the CICS Web service support will link to.

`REQMEM=value`
Specifies the name of the partitioned data set member which contains the high level language structure for the Web service request:
- For a service provider, the Web service request is the input to the application program
- For a service requester, the Web service request is the output from the application program

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

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

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

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

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

STRUCTURE=(*request*,*response*)
For C and C++ only, specifies the names of the high level structures contained in the partitioned data set members specified in the REQMEM and RESPMEM parameters:

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

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

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

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

SYNCONRETURN=**NO**|YES
specifies whether the remote Web service can issue a syncpoint.

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

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

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

Acceptable characters:
A-Z a-z 0-9 $ @ # _ < >

URI=*value*
In a service provider, this parameter specifies the relative URI that a client will to use to access the Web service. CICS uses the value specified when it

generates a URIMAP resource from the Web service binding file created by
DFHLS2WS: the parameter specifies the path component of the URI to which
the URIMAP definition applies.

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

| **Acceptable characters:** |
| --- |
| `A-Z a-z 0-9 $ @ #` |
#

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

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

WSDLCP=**LOCAL**│**UTF-8**
Specifies the code page that is used to generate the WSDL document.

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

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

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

DFHWS2LS: WSDL to high level language conversion

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

Job control statements for DFHWS2LS

JOB Initiates the job.

EXEC Specifies the procedure name (DFHWS2LS).

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

INPUT.SYSUT1 DD
> Specifies the input. The input parameters are usually specified in the input stream. However, they can be defined in a data set, or in a member of a partitioned data set.

Symbolic parameters

The following symbolic parameters are defined in cataloged procedure DFHWS2LS:

JAVADIR=*path*
> Specifies the name of the Java directory that is used by DFHWS2LS. The value of this parameter is appended to `/usr/lpp/` giving a complete path name of `/usr/lpp/`*path*.
>
> Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **JAVADIR** parameter.

\# **PATHPREF**=*prefix*
\# > Specifies an optional prefix that extends the HFS directory path used on other
\# > parameters. The default is the empty string.
\#
\# > Normally, you do not need to specify this parameter; the default value is the
\# > value that was supplied to the CICS installation job (DFHISTAR) in the **JAVADIR**
\# > parameter.

TMPDIR=*tmpdir*
> Specifies the location of a directory in HFS that DFHWS2LS uses as a temporary work space. The user ID under which the job runs must have read and write permission to this directory.
>
> The default value is `/tmp`.

TMPFILE=*tmpprefix*
> Specifies a prefix that DFHWS2LS uses to construct the names of the temporary workspace files.
>
> The default value is `WS2LS`.

USSDIR=*path*
> Specifies the name of the CICS TS directory in the UNIX system services HFS. The value of this parameter is appended to `/usr/lpp/cicsts/` giving a complete path name of `/usr/lpp/cicsts/`*path*.
>
> Normally, you do not need to specify this parameter; the default value is the value that was supplied to the CICS installation job (DFHISTAR) in the **USSDIR** parameter.

\# **SERVICE**=*value*
\# > Use this parameter only when directed to do so by IBM support.

The temporary work space

DFHWS2LS creates the following three temporary files during execution:
> *tmpdir*/*tmpprefix*.in
> *tmpdir*/*tmpprefix*.out
> *tmpdir*/*tmpprefix*.err

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

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

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

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

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

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

Input parameters for DFHWS2LS

If you need any help understanding this syntax diagram, see "Syntax notation" on page xii.

```
#                 ▶▶──PDSLIB=value──┬──────────────┬──┬──────────────┬──┬──────────────┬────────▶
                                    └─PDSCP=value──┘  └─REQMEM=value─┘  └─RESPMEM=value─┘

#   ▶──┬─LANG=COBOL──────────┬──────────────────────────────────────────────────────────────────▶
        ├─LANG=PLI-ENTERPRISE─┤
        ├─LANG=PLI-OTHER──────┤
        ├─LANG=C──────────────┤
        └─LANG=CPP────────────┴──┬─────────────────────────────────────────────────────┬─
                                 │          ┌─DFHREQUEST─┐    ┌─DFHRESPONSE─┐            │
                                 └─STRUCTURE=(─┼──────────┼─,─┼───────────┼─)─┘
                                              └─request──┘    └─response──┘

#   ▶──┬───────────────────────────────────────────────────────────────────────────────────────▶
        │                            ┌─PGMINT=CHANNEL──┐  ┌─CONTID=value─┐
        └─PGMNAME=value─┬───────────┬┼──────────────────┴──┴──────────────┘  ┬─────────────────┬  ┬───────────┬
                        └─URI=value─┘└─PGMINT=COMMAREA──────────────────────┘  └─TRANSACTION=name─┘  └─USERID=id─┘

#   ▶──WSBIND=value──WSDL=value──┬─MAPPING-LEVEL=1.0──────────────────────────┬──┬─MINIMUM-RUNTIME-LEVEL=MINIMUM─┬───▶
                                 ├─MAPPING-LEVEL=1.1──────────────────────────┤  ├─MINIMUM-RUNTIME-LEVEL=1.0─────┤
                                 └─MAPPING-LEVEL=1.2─┤ Advanced data mapping ├┘  ├─MINIMUM-RUNTIME-LEVEL=1.1─────┤
                                                                                 ├─MINIMUM-RUNTIME-LEVEL=1.2─────┤
                                                                                 └─MINIMUM-RUNTIME-LEVEL=CURRENT─┘

#   ▶──┬───────────────────────────────────────────┬──┬──────────────────────────┬──┬──────────────────────────┬──▶
        └─HTTPPROXY=─┬─domain name─┬─:port number─┘  └─HTTPPROXY-USERNAME=value─┘  └─HTTPPROXY-PASSWORD=value─┘
                     └─IP address──┘

#   ▶──┬───────────────┬──┬─────────────┬──┬─LOGFILE=value──────────────┬────────────────────────────────────────▶◀
        └─BINDING=value─┘  └─CCSID=value─┘  │                 ┌─NO─┐     │
                                            └─SYNCONRETURN=─┤    ├──────┘
                                              └─SYNCONRETURN=YES─┘
```

Advanced data mapping:

```
                            ┌─CHAR-VARYING-LIMIT=32767─┐  ┌─CHAR-MULTIPLIER=1──────┐
├──┬──────────────────────┬─┼──────────────────────────┴──┴────────────────────────┴────────▶
   ├─CHAR-VARYING=NO──────┤ └─CHAR-VARYING-LIMIT=value─┘  └─CHAR-MULTIPLIER=value─┘
   ├─CHAR-VARYING=NULL────┤
   └─CHAR-VARYING=YES─────┘

   ┌─DEFAULT-CHAR-MAXLENGTH=255─┐
▶──┼────────────────────────────┴──────────────────────────────────────────────┤
   └─DEFAULT-CHAR-MAXLENGTH=value─┘
```

Parameter use

- You can specify the input parameters in any order.
- Each parameter must start on a new line.

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

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

is equivalent to

```
WSBIND=wsbinddir/app1
```
- A # character in the first character position of the line is a comment character. The line is ignored.

Parameter descriptions

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

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

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

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

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

CHAR-VARYING=**NO**│**NULL**│**YES**
> Specifies how variable length character data is mapped. Variable length character data is where the minimum and maximum length of a field is different. This parameter can only be used when the mapping level is 1.2. If you do not specify this parameter, the default mapping depends on the language that is specified. These defaults are described in the mappings for each language and XML schema in High level language and XML schema mapping. The options that you can select are:

> **NO**   Variable length character data is mapped as fixed length strings.

> **NULL**   Variable length character data is mapped to null terminated strings.

> **YES**   Variable length character data is mapped to a CHAR VARYING data type in PL/I. In the COBOL, C and C++ languages, variable length character data is mapped to an equivalent representation that comprises of two related elements - data length and the data.

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

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

CONTID=*value*

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

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.

DEFAULT-CHAR-MAXLENGTH=<u>255</u>|*value*
Specifies the default field length of character data in characters for mappings
where no length is implied in the Web service description document. The *value*
of this parameter can be a positive integer in the range of 1 to 2147483647.

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

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

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

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

In other cases, this parameter is not required.

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

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

LANG=**COBOL**

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

LANG=**PLI-ENTERPRISE**

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

LANG=**PLI-OTHER**
Specifies that the programming language of the high level language structure is
a level of PL/I other than Enterprise PL/I.

LANG=**C**

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

LANG=**CPP**

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

LOGFILE=*value*

> The fully qualified HFS name of the file into which DFHWS2LS writes its activity log and trace information. DFHWS2LS creates the file (but not the directory structure) if it does not already exist.
>
> Normally you will not need to use this file, but it may be requested by the IBM service organization if you encounter problems with DFHWS2LS.

MAPPING-LEVEL={**1.0**|**1.1**|**1.2**}

> Specifies the level of mapping that DFHWS2LS should use when generating the Web service binding file and language structure. This parameter is available when you apply APAR PK15904. You also need to apply APAR PK23547 if you want to use the 1.2 mapping level option. The options you can select are:
>
> **1.0**   This is the default mapping level.
>
> **1.1**   XML attributes, `<list>` data types, and `<union>` data types are mapped to the language structure. Character and binary data that has a maximum length of more than 32,767 bytes is mapped to a container. The container name is created in the language structure.
>
> **1.2**   Use the parameters **CHAR-VARYING** and **CHAR-VARYING-LIMIT** to control how character data is mapped and processed at run time. If you do not specify either of these parameters, binary and character data that has a maximum length less than 32768 bytes is mapped to a VARYING structure for all languages except C++, where character data is mapped to a null terminated string.
>
> For details of what is supported at each level of mapping, see Mapping levels for the CICS Web services assistant.

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

> Specifies the minimum CICS runtime environment that the Web service binding file can be deployed into. If you select a level that does not match the other parameters that you have specified, you receive an error message. The options you can select are:
>
> **MINIMUM**
>
> > The lowest possible runtime level of CICS is allocated automatically given the parameters that you have specified.
>
> **1.0**   The generated Web service binding file deploys successfully into a CICS TS 3.1 region that does not have APARs PK15904 and PK23547 applied. You cannot specify the **CCSID** or **MAPPING-LEVEL** parameter, or any other optional parameters that rely on the **MAPPING-LEVEL** parameter.
>
> **1.1**   The generated Web service binding file deploys successfully into a CICS TS 3.1 region that has at least APAR PK15904 applied. You cannot specify the **CCSID** parameter or use a mapping level of 1.2 for the **MAPPING-LEVEL** parameter. You cannot specify any optional parameters that rely on the 1.2 level of mapping.
>
> **1.2**   The generated Web service binding file deploys successfully into a CICS TS 3.1 region that has both APAR PK15904 and PK23547 applied. You can use any optional parameter at this level.
>
> **CURRENT**
>
> > The generated Web service binding file deploys successfully into a CICS region at the same runtime level as the one you are using to generate the Web service binding file.

PDSLIB=*value*

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

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

PGMINT=**CHANNEL**|**COMMAREA**

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

CHANNEL

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

COMMAREA

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

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

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*

This parameter specifies the name of a CICS program.

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

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

REQMEM=*value*

Specifies a 1 - 6 character prefix that DFHWS2LS uses to generate the names of the partitioned data set members that will contain the high level language structures for the Web service request:
• For a service provider, the Web service request is the input to the application program
• For a service requester, the Web service request is the output from the application program

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

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

RESPMEM=*value*
>   Specifies a 1 - 6 character prefix that DFHWS2LS uses to generate the names
>   of the partitioned data set members that will contain the high level language
>   structures for the Web service response:
>   - For a service provider, the Web service response is the output from the
>   application program
>   - For a service requester, the Web service response is the input to the
>   application program
>
>   DFHWS2LS generates a partitioned data set member for each operation. It
>   generates the member name by appending a two digit number to the prefix.
>
>   If there is no response (that is, for one way messages) omit this parameter.

STRUCTURE=(*request*,*response*)
>   For C and C++ only, specifies how the names of the request and response
>   structures are generated.
>
>   The generated request and response structures are given names of *requestnn*
>   and *responsenn* where *nn* is a numeric suffix that is generated to distinguish the
>   structures for each operation.
>   If one or both names is omitted, the structures have the same name as the
>   partitioned data set member names generated from the REQMEM and
>   RESPMEM parameters that you specify.

SYNCONRETURN=<u>NO</u>|YES
>   specifies whether the remote Web service can issue a syncpoint.
>
>   **NO**   The remote Web service cannot issue a syncpoint. This value is the
>   default. If the remote Web service issues a syncpoint, it fails with an
>   ADPL abend.
>
>   **YES**   The remote Web service can issue a syncpoint. If you select YES, the
>   remote task is committed as a separate unit of work when control
>   returns from the remote Web service. If the remote Web service
>   updates a recoverable resource and a failure occurs after it returns, the
>   update to that resource cannot be backed out.

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

> | **Acceptable characters:** |
> | --- |
> | A-Z a-z 0-9 $ @ # _ < > |

URI=*value*
>   In a service provider, this parameter specifies the relative URI that a client will
>   use to access the Web service. CICS uses the value specified when it
>   generates a URIMAP resource from the Web service binding file created by
>   DFHWS2LS: the parameter specifies the path component of the URI to which
>   the URIMAP definition applies.
>
>   In a service requester, the URI of the target Web service is **not** specified with
>   this parameter: the URI specified in the Web service description is used,
>   although you can override that with the URI option on the **EXEC CICS INVOKE
>   WEBSERVICE** command.

USERID=*id*
>   In a service provider, this parameter specifies a 1-8 character user ID which can

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

<table>
<tr><td>Acceptable characters:</td></tr>
<tr><td><code>A-Z a-z 0-9 $ @ #</code></td></tr>
</table>

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

WSDL=*value*
The fully qualified HFS name of the file that contains the Web service
description.

Other information

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

The pipeline configuration file

The configuration of a pipeline used to handle a Web service request is specified in
an XML document, known as a *pipeline configuration file*.

The pipeline configuration file is stored in the z/OS UNIX System Services
hierarchical file system (HFS), and its name is specified in the CONFIGFILE
attribute of a PIPELINE resource definition. Use a suitable XML editor or text editor
to work with your pipeline configuration files. When you work with configuration files,
ensure that the character set encoding is US EBCDIC (Code page 037).

When CICS processes a Web service request, it uses a pipeline of one or more
message handlers to handle the request. A pipeline is configured to provide aspects
of the execution environment that apply to different categories of applications, such
as support for Web Service Security, and Web Service transactions. Typically, a
CICS region that has a large number of service provider or service requester
applications will need several different pipeline configurations. However, where
different applications have similar requirements, they can share the same pipeline
configuration.

There are two kinds of pipeline configuration: one describes the configuration of a
service provider pipeline; the other describes a service requester pipeline. Each is
defined by its own schema, and each has a different root element.

Pipeline	Schema	Root element
Service provider	`Provider.xsd`	`<provider_pipeline>`
Service requester	`Requester.xsd`	`<requester_pipeline>`

Although many of the XML elements used are common to both kinds of pipeline
configuration, others are used only in one or the other, so you cannot use the same
configuration file for both a provider and requester.

The immediate sub-elements of the `<provider_pipeline>` and `<requester_pipeline>` elements are:

• A `<service>` element, which specifies the message handlers that are invoked for
every request. This element is mandatory when used within the
`<provider_pipeline>` element, and optional within the `<requester_pipeline>`
element.

• An optional `<transport>` element, which specifies message handlers that are
selected at run time, based upon the resources that are being used for the
message transport.

• For the `<provider_pipeline>` only, an `<apphandler>` element, which is used in
some cases to specify the target application (or wrapper program) that provides
the service.

• An optional `<service_parameter_list>` element, which contains the parameters
that are available to the message handlers in the pipeline.

Associated with the pipeline configuration file is a PIPELINE resource. The
attributes include CONFIGFILE, which specifies the name of the pipeline
configuration file in HFS. When you install a PIPELINE definition, CICS reads the
information that it needs in order to configure the pipeline from the file.

CICS supplies sample configuration files that you can use as a basis for developing
your own. They are provided in library `/usr/lpp/cics/samples/pipelines`.

File   Description

basicsoap11provider.xml
> A pipeline definition for a service provider that uses the CICS-provided
> SOAP 1.1 handler, for use when the application has been deployed using
> the CICS Web services assistant.

basicsoap11requester.xml
> A pipeline definition for a service requester that uses the CICS-provided
> SOAP 1.1 handler, for use when the application has been deployed using
> the CICS Web services assistant.

wsatprovider.xml
> A pipeline definition that adds configuration information for Web Services
> transactions to `basicsoap11provider.xml`.

wsatrequester.xml
> A pipeline definition that adds configuration information for Web Services
> transactions to `basicsoap11requester.xml`.

Example pipeline configuration file

This is a simple example of a configuration file for a service provider pipeline:

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline/provider.xsd">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The pipeline contains just one message handler, the CICS-supplied SOAP 1.1
message handler. The handler links to program DFHPITP.

- The `<provider_pipeline>` element is the root element of the pipeline configuration file for a service provider pipeline.
- The `<service>` element specifies the message handlers that are invoked for every request. In the example, there is just one message handler.
- The `<terminal_handler>` element contains the definition of the terminal message handler of the pipeline.
- The `<cics_soap_1.1_handler>` indicates that the terminal handler of the pipeline is the CICS-supplied handler program for SOAP 1.1 messages.
- The `<apphandler>` element specifies the name of the program to which the terminal handler of the pipeline will link by default. In this case, the program is DFHPITP, which is the CICS-supplied target program for applications deployed with the CICS Web services assistant. For programs that are not deployed with the Web services assistant, this is the name of the target application program.

Changes to CICS externals

Changes to resource definition

Web services in CICS uses three new CICS resources: PIPELINE, URIMAP, and WEBSERVICE.

PIPELINE
> A PIPELINE resource definition specifies the processing to be applied to a Web service request. For more information see "PIPELINE resource definitions" on page 56.

> The PIPELINE refers to an XML file which defines the processing nodes.

URIMAP
> URIMAP definitions enable CICS to match the URIs of requests from Web clients, or requests to a remote server, and provide information on how to process the requests. For more information see "URIMAP resource definitions" on page 135.

> The following attributes of the URIMAP resource are relevant:

> **PIPELINE**
>> Specifies the PIPELINE resource definition that provides information about the message handlers which will act on the service request from the client.

> **WEBSERVICE**
>> Specifies the WEBSERVICE that defines aspects of the run time environment for a CICS application used in a Web services setting.

> **TRANSACTION**
>> Specifies the name of an alias transaction that is used to start the pipeline.

> **USERID**
>> Specifies the user ID under which the alias transaction is attached.

WEBSERVICE
> A WEBSERVICE resource defines aspects of the run time environment for a CICS application program deployed in a Web services setting, where the mapping between application data structure and SOAP messages has been generated using the CICS Web services assistant. For more information, see "WEBSERVICE resource definitions" on page 58.

PIPELINE resource definitions

A PIPELINE resource definition is used when a CICS application is in the role of a Web service provider or requester. It provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE definition defines an infrastructure that can be used by many applications.

The information about the processing nodes is supplied indirectly: the PIPELINE specifies the name of an HFS file which contains an XML description of the nodes and their configuration.

An inbound Web service request (that is, a request by which a client invokes a Web service in CICS) is associated with a PIPELINE resource by the URIMAP resource. The URIMAP identifies the PIPELINE resource that applies to the URI associated with the request; the PIPELINE specifes the processing that is to be performed on the message.

PIPELINE attributes:

```
►►──PIPELINE(name)──GROUP(groupname)────────────────────────────────►
                                    └─DESCRIPTION(text)─┘

       ┌─SHELF(/var/cicsts)─┐  ┌─STATUS(ENABLED)──┐
►──CONFIGFILE(name)─┤                    ├──┤                  ├───────►
                    └─SHELF(directory)───┘  └─STATUS(DISABLED)─┘

►───────────────────────────────────────────────────────────────────►◄
    └─WSDIR(directory)─┘
```
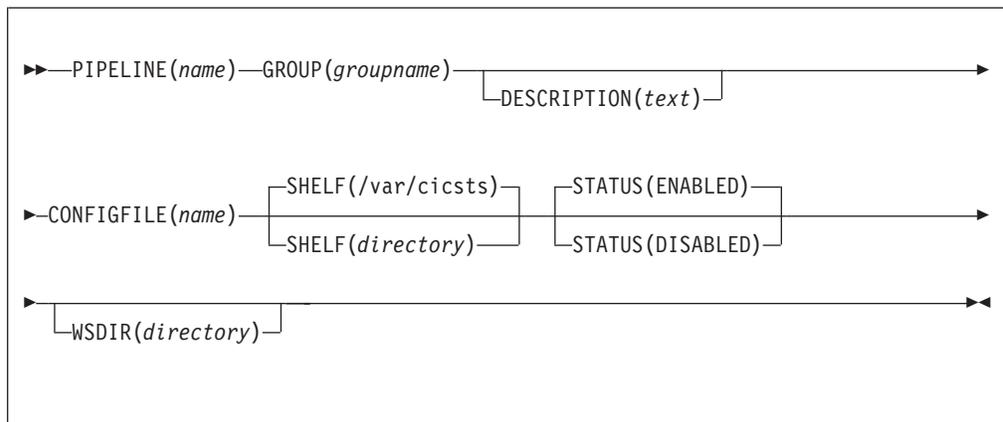
PIPELINE(*name*)
> Specifies the name of this PIPELINE. The name can be up to eight characters in length.

> | **Acceptable characters:** |
> | --- |
> | A-Z 0-9 $ @ # |
> | |
> | Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase. |

GROUP(*groupname*)
> Every resource definition must have a GROUP name. The resource definition becomes a member of the group and is installed in the CICS system when the group is installed.

> | **Acceptable characters:** |
> | --- |
> | A-Z 0-9 $ @ # |
> | |
> | Any lower case characters you enter are converted to upper case. |

The GROUP name can be up to eight characters in length. Lowercase characters are treated as uppercase characters. Do not use group names beginning with DFH, because these characters are reserved for use by CICS.

DESCRIPTION(*text*)
You can provide a description of the resource you are defining in this field. The description text can be up to 58 characters in length. There are no restrictions on the characters that you can use. However, if you use parentheses, ensure that for each left parenthesis there is a matching right one. If you use the CREATE command, for each single apostrophe in the text, code two apostrophes.

CONFIGFILE(*name*)
Specifies the name of an HFS file that contains information about the processing nodes that will act on a service request, and on the response.

Acceptable characters:
A-Z a-z 0-9 . / _ # @

The value specified must be a valid name for an HFS file:
- The name must not contain imbedded space characters
- The name must not contain consecutive instances of the / character

The name is case-sensitive.

SHELF({**/var/cicsts/**|*directory*})
Specifies the 1–255 character fully-qualified name of a directory (a *shelf*, primarily for Web service binding files) on HFS.

Acceptable characters:
A-Z a-z 0-9 . / _ # @

The value specified must be a valid name for an HFS file:
- The name must not contain imbedded space characters
- The name must not contain consecutive instances of the / character

The name is case-sensitive.

CICS regions into which the PIPELINE definition is installed must have full permissions to the shelf directory—read, write, and the ability to create subdirectories.

A single shelf directory can be shared by multiple CICS regions and by multiple PIPELINE definitions. Within a shelf directory, each CICS region uses a separate subdirectory to keep its files separate from those of other CICS regions. Within each region's directory, each PIPELINE uses a separate subdirectory.

After a CICS region performs a cold or initial start, it deletes its subdirectories from the shelf before trying to use the shelf.

You should not attempt to modify the contents of a shelf that is referred to by an installed PIPELINE definition. If you do, the effects are unpredictable.

STATUS({**ENABLED**|**DISABLED**})
Specifies the initial status of the PIPELINE when it is installed:

ENABLED
Web service requests for this PIPELINE are processed normally.

DISABLED
>   Web service requests for this PIPELINE cannot be processed.

WSDIR(*directory*)
>   specifies the 1–255 character fully-qualified name of the *Web service binding directory* (also known as the *pickup directory*) on HFS.

> | **Acceptable characters:** |
> | --- |
> | A-Z a-z 0-9 . / _ # @ |

>   The value specified must be a valid name for an HFS file:
>   - The name must not contain imbedded space characters
>   - The name must not contain consecutive instances of the / character
>
>   The name is case-sensitive.
>
>   The Web service binding directory contains Web service binding files that are associated with a PIPELINE, and that are to be installed automatically by the CICS scanning mechanism. When the PIPELINE definition is installed, CICS scans the directory and automatically installs any Web service binding files it finds there. Note that this happens regardless of whether the PIPELINE is installed in enabled or disabled state.
>
>   If you specify a value for the WSDIR attribute, it must refer to a valid HFS directory to which the CICS region has at least read access. If this is not the case, any attempt to install the PIPELINE resource will fail.
>
>   If you do not specify a value for WSDIR, no automatic scan takes place on installation of the PIPELINE, and PERFORM PIPELINE SCAN commands will fail.

WEBSERVICE resource definitions

A WEBSERVICE resource defines aspects of the run time environment for a CICS application program deployed in a Web services setting, where the mapping between application data structure and SOAP messages has been generated using the CICS Web services assistant. Although CICS provides the usual resource definition mechanisms for WEBSERVICE resources, they are typically installed dynamically, using the output produced by the assistant.

The aspects of the run time environment that are defined by the WEBSERVICE resource are:

A pipeline
>   Defines the set of message handlers that operate on Web service requests and responses. The WEBSERVICE resource specifies a separate PIPELINE resource which, in turn, specifies the pipeline configuration file.

A Web service binding file
>   Contains information that is used at run time to perform the mapping between application data structures and SOAP messages. The Web service binding file is generated by the CICS-supplied tools.

A Web service description
>   The Web services description is used only when runtime validation of SOAP messages is required. Validation of each message is performed against its schema, which is imbedded within the Web service description.

An inbound Web service request (that is, a request by which a client invokes a Web service in CICS) is associated with a WEBSERVICE resource by the URIMAP

resource. The URIMAP identifies the WEBSERVICE resource that applies to the URI in the inbound message; the WEBSERVICE specifies the processing that is to be performed on the message.

Although CICS provides the usual resource definition mechanisms for creating WEBSERVICE resources, and installing them in your CICS region, you can instead use the scanning mechanism to dynamically install WEBSERVICE resources in your running CICS system. The advantages of this approach are that it reduces the amount of resource definition required, and that CICS can make direct use of information provided at development time.

To invoke the scanning mechanism, use the **PERFORM PIPELINE** command.

The name of a dynamically-installed WEBSERVICE is derived from the name of the Web service binding file from which the WEBSERVICE definition is generated, and has a maximum length of 32 characters; the names of WEBSERVICE definitions installed from the CSD or with the **EXEC CICS CREATE WEBSERVICE** are limited to eight characters. For example, a Webservice binding file whose HFS name is `/samples/Webservices/WSDir/InquireSingle.wsbind` generates a WEBSERVICE definition named `InquireSingle`

WEBSERVICE attributes:

```
>>--WEBSERVICE(name)--GROUP(groupname)--------------------------->
                                       |-DESCRIPTION(text)-|

 >-PIPELINE(pipelinename)--WSBIND(hfsfile)--|-VALIDATION(NO)--|--->
                                            |-VALIDATION(YES)-|

 >--------------------------------------------------------------><
   |-WSDLFILE(hfsfile)-|
```

WEBSERVICE(*name*)

Specifies the 1-8 character name of the WEBSERVICE.

Acceptable characters:
A-Z a-z 0-9 $ @ # . / - _ % & ¢ ? ! : | " = ¬ , ; < >

Do not use names beginning with DFH, because these characters are reserved for use by CICS.

GROUP(*groupname*)

Every resource definition must have a GROUP name. The resource definition becomes a member of the group and is installed in the CICS system when the group is installed.

Acceptable characters:
A-Z 0-9 $ @ #

Any lower case characters you enter are converted to upper case.

The GROUP name can be up to eight characters in length. Lowercase characters are treated as uppercase characters. Do not use group names beginning with DFH, because these characters are reserved for use by CICS.

DESCRIPTION(*text*)
You can provide a description of the resource you are defining in this field. The description text can be up to 58 characters in length. There are no restrictions on the characters that you can use. However, if you use parentheses, ensure that for each left parenthesis there is a matching right one. If you use the CREATE command, for each single apostrophe in the text, code two apostrophes.

PIPELINE(*pipelinename*)
Specifies the 1-8 character name of the PIPELINE with which this WEBSERVICE is associated.

Acceptable characters:
A-Z 0-9 $ @ #
Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase.

VALIDATION(NO│YES)
Specifies whether full validation of SOAP messages against the corresponding schema in the Web service description should be performed at run time. Validating a SOAP message against its schema incurs considerable processing overhead, and you should normally specify VALIDATION(NO).

Full validation ensures that all SOAP messages that are sent and received are valid XML with respect to the XML schema. If VALIDATION(NO) is specified, sufficient validation is performed to ensure that the message contains well-formed XML.

WSBIND(*hfsfile*)
Specifies the 1-255 character fully-qualified file name of the Web service binding file on HFS.

\#
\#
\#

Acceptable characters:
A-Z a-z 0-9 . / _ # @

The name is case-sensitive, and cannot contain spaces. The name must not end with a / character, and must not contain consecutive instances of the / character.

WSDLFILE(*hfsfile*)
Specifies the 1-255 character fully-qualified file name of the Web service description (WSDL) file on HFS. This file is used when full runtime validation is active.

\#
\#
\#

Acceptable characters:
A-Z a-z 0-9 . / _ # @

The name is case-sensitive, and cannot contain spaces. The name must not end with a / character, and must not contain consecutive instances of the / character.

Changes to the application programming interface

There is a new command that enables a CICS application program to invoke a Web service. For more details, see "INVOKE WEBSERVICE" on page 61.

There are three new commands that enable a SOAP node to construct a SOAP fault. For more details, see:

- "SOAPFAULT ADD" on page 62
- "SOAPFAULT CREATE" on page 63
- "SOAPFAULT DELETE" on page 65

INVOKE WEBSERVICE

This command invokes a Web service from a CICS application. The command specifies the name of a WEBSERVICE resource, which contains information about the service to be invoked.

INVOKE WEBSERVICE

```
►►──INVOKE─WEBSERVICE(name)──CHANNEL(name)──OPERATION(data-area)──────────────►

►─┬──────────────────────┬──────────────────────────────────────────────────►◄
  └──URI(data-area)───────┘
```

This command is threadsafe.
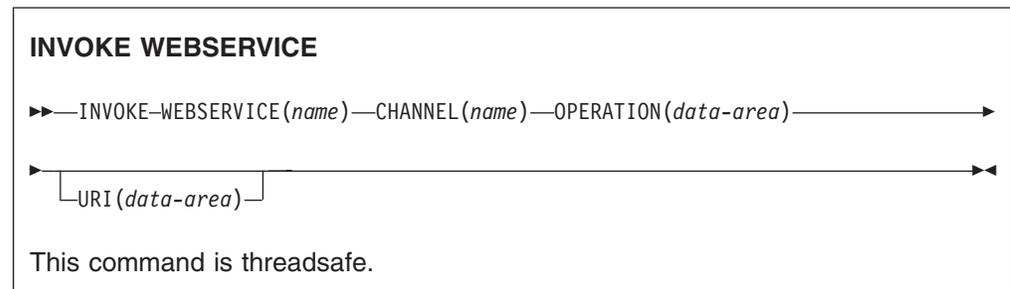
Options

CHANNEL(*name*)

specifies the name of the channel used to pass the containers that hold the data mapped by the application data structure. On return, the same channel holds the response from the Web service, again mapped by the application data structure. The name of the channel can be up to 16 characters. If *name* is a variable, and it contains a name that is less than 16 characters, then the variable must be padded with trailing blanks.

OPERATION(*data-area*)

specifies a data area containing the name of the operation that is to be invoked. The name of the operation is contained in the WSDL for the target Web service. The data area must be 255 characters long; if the operation name is less than 255 characters, then the data area must be padded with trailing blanks.

URI(*data-area*)

specifies a data area containing the URI of the Web service to be invoked. If specified, this option supersedes any URI specified in the WEBSERVICE resource definition. If this option is omitted, then the WEBSERVICE resource definition must include either a provider URI or a provider application name. The data area must be 255 characters long; if the URI is less than 255 characters, then the data area must be padded with trailing blanks.

WEBSERVICE(*name*)

specifies 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. The name of the WEBSERVICE can be up to 32 characters. If *name* is a variable, and it contains a name that is less than 32 characters, then the variable must be padded with trailing blanks.

SOAPFAULT ADD

This command adds information to an existing SOAPFAULT object. You can use this command only in a program that is invoked from a CICS-supplied SOAP message handler.

```
SOAPFAULT ADD

►►──SOAPFAULT ADD──────────────────────────────────────────────────────────────►

                                                              ┌─NATLANG('en')────┐
►──┬─FAULTSTRING(data-value)──FAULTSTRLEN(data-value)─┬────────┼──────────────────┼──►
   └─SUBCODESTR(data-value)──SUBCODELEN(data-value)───┘        └─NATLANG(data-value)─┘

►──┬──────────────────────┬────────────────────────────────────────────────────►◄
   └─FROMCCSID(data-value)─┘
```

This command is threadsafe.
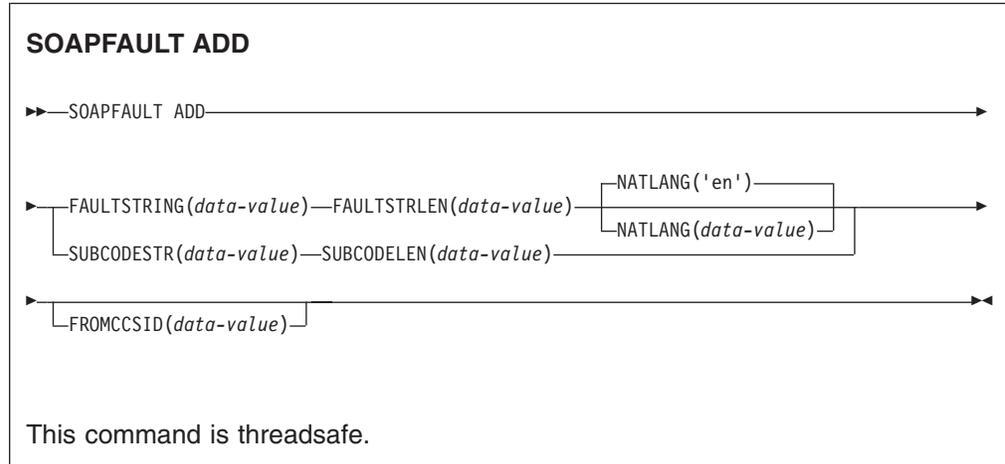
Options

SUBCODESTR(*data-value*)
Specifies the contents of a `<Subcode>` element that is to be added to the
SOAPFAULT object. The subcode can be up to 64 characters in length, and
must be an XML qualified name (`QName`). An XML qualified name has the form
prefix:name.
• For SOAP 1.1, this option is ignored.
• For SOAP 1.2, this option supplies the contents of the `<Subcode>` element.

SUBCODELEN(*data-value*)
specifies the length, as a fullword binary value, of the `<Subcode>` element specified in the `SUBCODESTR` option.

FAULTSTRING(*data-value*)
specifies a human-readable explanation of the fault. The FAULTSTRING can be up to 2056 characters in length.
• For SOAP 1.1, this option supplies the contents of the `<faultstring>` element.
• For SOAP 1.2, this option supplies the contents of the `<Reason>` element.

FAULTSTRLEN(*data-value*)
Specifies the length, as a fullword binary value, of the `FAULTSTRING` option.

FROMCCSID(*data-value*)
Specifies, as a fullword decimal number, the current Coded Character Set Identifier (CCSID) of the character data to be put into the SOAP fault. If this option is not specified, CICS uses the value which is specified in the LOCALCCSID system initialization parameter. For more information about CCSIDs, and a list of the CCSIDs supported by CICS, see *CICS Family: Communicating from CICS on System/390*.

NATLANG(*data-value*)
Specifies an eight character field containing the national language used for the `FAULTSTRING`. The language is specified using the XML 1.0 language identification. The default value is `'en'` (English).

When the language identifier is shorter than eight characters, you must pad it on the right with space characters in the character set specified in the FROMCCSID option (or the CICS LOCALCCSID). For example, if you specify the UTF-8 character set with FROMCCSID(1208), you must pad the NATLANG value with `X'20'` characters.

SOAPFAULT CREATE

This command creates a SOAP fault. You can use this command only in a program that is invoked from a CICS-supplied SOAP message handler.

SOAPFAULT CREATE

```
►►──SOAPFAULT CREATE──┬─FAULTCODE(cvda)─────────────────────────────┬──────►
                      ├─CLIENT──────────────────────────────────────┤
                      ├─SERVER──────────────────────────────────────┤
                      ├─SENDER──────────────────────────────────────┤
                      ├─RECEIVER────────────────────────────────────┤
                      └─FAULTCODESTR(data-value)──FAULTCODELEN(data-value)─┘

                                                      ┌─NATLANG('en')─────┐
►──FAULTSTRING(data-value)──FAULTSTRLEN(data-value)──┼───────────────────┼──►
                                                      └─NATLANG(data-value)┘

►──┬──────────────────────────────────────────┬───────────────────────────►
   └─ROLE(data-value)──ROLELENGTH(data-value)─┘

►──┬──────────────────────────────────────────────┬───────────────────────►
   └─FAULTACTOR(data-value)──FAULTACTLEN(data-value)─┘

►──┬───────────────────────────────────────────┬──┬────────────────────────┬──►◄
   └─DETAIL(data-value)──DETAILLENGTH(data-value)─┘  └─FROMCCSID(data-value)─┘
```

This command is threadsafe.
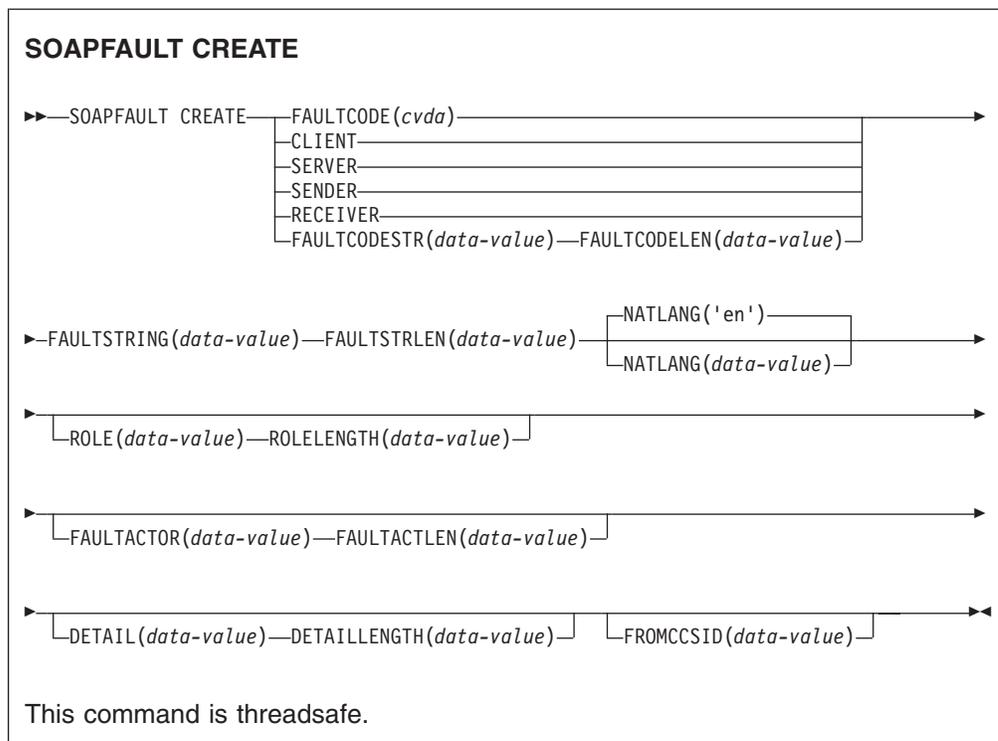
Options

DETAIL(*data-value*)
- For SOAP 1.1, this option supplies the contents of the `<detail>` element of the SOAP fault.
- For SOAP 1.2, this option supplies the contents of the `<Detail>` element of the SOAP fault..

It should contain either one or more valid namespace-qualified XML elements, or whitespace. Refer to the appropriate SOAP specifications for a full description of the valid content of the element.

The element carries application-specific error information related to the `<Body>` element, and is used when the contents of the `<Body>` element could not be successfully processed. For SOAP 1.1, the `<detail>` element must be present if the contents of the `<Body>` element could not be successfully processed; for SOAP 1.2, the `<Detail>` element is optional.

If the SOAPFAULT CREATE command is issued in a header handler program the `<detail>` or `<Detail>` element is carried in a header block.

DETAILLENGTH(*data-value*)
specifies the length, as a fullword binary value, of the `DETAIL` option.

FAULTACTLEN(*data-value*)
    Specifies the length, as a fullword binary value, of the FAULTACTOR option.

FAULTACTOR(*data-value*)
- For SOAP 1.1, this option supplies the contents of the `<faultactor>` element.
- For SOAP 1.2, this option supplies the contents of the `<Node>` element.

The FAULTACTOR option can be up to 2056 characters in length, and must be a valid URI (`anyURI`).

FAULTCODE(*cvda*)

  CLIENT
  SENDER
    For SOAP 1.1 specifies a SOAP Fault code of `Client`

    For SOAP 1.2 specifies a SOAP Fault code of `Sender`

  SERVER
  RECEIVER
    For SOAP 1.1 specifies a SOAP Fault code of `Server`

    For SOAP 1.2 specifies a SOAP Fault code of `Receiver`

FAULTCODELEN(*data-value*)
    Specifies the length, as a fullword binary value, of the FAULTCODESTR option.

FAULTCODESTR(*data-value*)
    Specifies a user-defined SOAP Fault code. The Fault code can be up to 64 characters in length, and must be an XML qualified name (`QName`). The use of the "." (dot) character to separate Fault code values is not supported.
- For SOAP 1.1, this option supplies the contents of the `<faultcode>` element.
- For SOAP 1.2, this option supplies the contents of the `<Code>` element.

FAULTSTRING(*data-value*)
    specifies a human-readable explanation of the fault. The FAULTSTRING can be up to 2056 characters in length.
- For SOAP 1.1, this option supplies the contents of the `<faultstring>` element.
- For SOAP 1.2, this option supplies the contents of the `<Reason>` element.

FAULTSTRLEN(*data-value*)
    Specifies the length, as a fullword binary value, of the FAULTSTRING option.

FROMCCSID(*data-value*)
    Specifies, as a fullword decimal number, the current Coded Character Set Identifier (CCSID) of the character data to be put into the SOAP fault. If this option is not specified, CICS uses the value which is specified in the LOCALCCSID system initialization parameter. For more information about CCSIDs, and a list of the CCSIDs supported by CICS, see *CICS Family: Communicating from CICS on System/390*.

NATLANG(*data-value*)
    Specifies an eight character field containing the national language used for the FAULTSTRING. The language is specified using the XML 1.0 language identification. The default value is `'en'` (English).

When the language identifier is shorter than eight characters, you must pad it on the right with space characters in the character set specified in the FROMCCSID option (or the CICS LOCALCCSID). For example, if you specify the UTF-8 character set with FROMCCSID(1208), you must pad the NATLANG value with `X'20'` characters.

ROLE(*data-value*)
> Specifies the URI that describes the role of the SOAP node that generated the fault. The `ROLE` option can be up to 2056 characters in length, and must be a valid URI (XML type `anyURI`).
> - For SOAP 1.1, this option is ignored.
> - For SOAP 1.2, this option supplies the contents of the `<Role>` element.

ROLELENGTH(*data-value*)
> Specifies the length, as a fullword binary value, of the `ROLE` option.

SOAPFAULT DELETE

This command deletes an existing SOAPFAULT object. You can use it only in a program that is invoked from a CICS-supplied SOAP message handler.

SOAPFAULT DELETE

►►—SOAPFAULT DELETE————————————————————————►◄

This command is threadsafe.

Changes to the system programming interface

CREATE PIPELINE command

Use the **CREATE PIPELINE** command to dynamically create a PIPELINE in your CICS region. The attributes you can specify on this command are described in "PIPELINE attributes" on page 56.

CREATE WEBSERVICE command

Use the **CREATE WEBSERVICE** command to dynamically create a WEBSERVICE in your CICS region. The attributes you can specify on this command are described in "WEBSERVICE attributes" on page 59.

DISCARD PIPELINE command

Use the **DISCARD PIPELINE** pipeline to remove a PIPELINE from your CICS region. The PIPELINE must be disabled before it can be discarded.

DISCARD WEBSERVICE command

Use the **DISCARD WEBSERVICE** command to remove a WEBSERVICE from your CICS region. The WEBSERVICE must be disabled before it can be discarded.

INQUIRE PIPELINE command

Use the **INQUIRE PIPELINE** to retrieve information about an installed PIPELINE.

```
┌──────────────────────────────────────────────────────────────────────┐
│ INQUIRE PIPELINE                                                       │
│                                                                        │
│                                  ┌─────────────────────────────────┐   │
│                                  │                                 │   │
│ ▶▶─INQUIRE─PIPELINE(data-value)──┴─┬─CONFIGFILE(data-area)─┬───────┴──▶◀│
│                                    ├─ENABLESTATUS(cvda)────┤           │
│                                    ├─SHELF(data-area)──────┤           │
│                                    └─WSDIR(data-area)──────┘           │
│                                                                        │
└──────────────────────────────────────────────────────────────────────┘
```
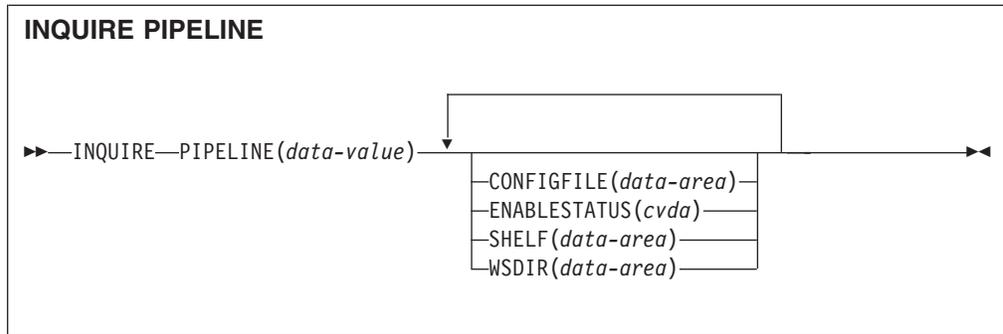
You can browse through all the PIPELINEs installed in your system by using the browse options (`START`, `NEXT`, and `END`) on **INQUIRE PIPELINE** commands.

Options

CONFIGFILE(*data-area*)
Returns the name of the pipeline configuration file associated with the PIPELINE resource. The name can be up to 255 characters long.

ENABLESTATUS(*cvda*)
Returns the status of the PIPELINE:

ENABLED
The PIPELINE is ready for use.

DISABLED
The PIPELINE is not processing requests, and is unable to accept new work. It may have failed to initialize, or may have been explicitly disabled.

ENABLING
The PIPELINE is being initialized; it is not yet ready to accept work.

DISABLING
The PIPELINE is quiescing before entering DISABLED state. It is not accepting new work, but is allowing currently-executing work to complete.

DISCARDING
A DISCARD command has been issued for the PIPELINE. The PIPELINE is quiescing before being discarded. It is not accepting new work, but is allowing currently-executing work to complete.

PIPELINE(*data-value*)
Specifies the name of the PIPELINE about which you are inquiring. The name can be up to 8 characters long.

SHELF(*data-area*)
Returns the name of the *shelf directory*. The name can be up to 255 characters long.

WSDIR(*data-area*)
Returns the name of the *Web service binding directory* (also known as the *pickup directory*). The name can be up to 255 characters long.

INQUIRE WORKREQUEST command
There is a new value for the `WORKTYPE` option on the **INQUIRE WORKREQUEST** command:

SOAP
Specifies that the work is being executed for a Web service request.

PERFORM PIPELINE command

Use the **PERFORM PIPELINE** command to initiate a scan of the Web service binding files that are associated with a PIPELINE.

```
PERFORM PIPELINE

>>—PERFORM—PIPELINE(name)—┬────────────────┬—————————————————><
                          ├—ACTION(cvda)—┤
                          └—SCAN————————┘
```
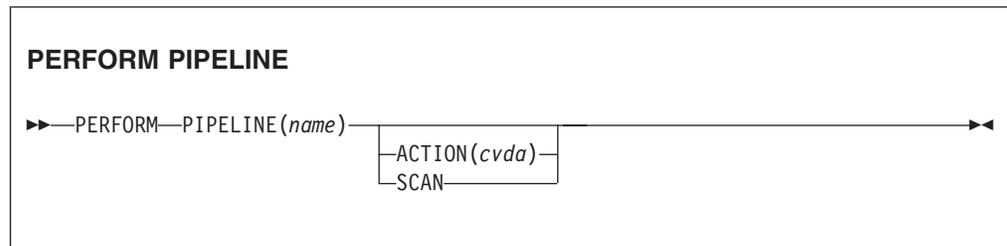
The Web service binding files that are scanned are located in the directory that is specified in the WSBIND attribute of the PIPELINE definition. If the WSBIND attribute is not specified, there is nothing to scan, and control returns to your program.

If the directory location specified is valid, CICS examines the Web service binding files in the directory to determine if they should be installed into the system:
- CICS installs any files it finds that have not been installed already.
- If a file has been installed already, but the file in the directory is newer than the one currently in use, the one that is in use is discarded, and the newer file is installed in its place.

If, for any reason, CICS fails to install an individual Web service binding file, processing continues with the remaining files in the directory. When the scan completes, the PIPELINE is available for use with whichever of the binding files were installed successfully.

Options

ACTION(*cvda*)
> Specifies a CVDA value indicating the action to be taken on the PIPELINE. CVDA values are:
>
> **SCAN** Scan the PIPELINE's Web service binding directory

PIPELINE(*name*)
> Specifies the name of the PIPELINE.

PERFORM STATISTICS RECORD command

This command supports the following new options:

PIPELINE
> Records statistics related to a PIPELINE. This includes information about HFS files.

SET PIPELINE command

Use the **SET PIPELINE** command to change the status of an installed PIPELINE:

PIPELINE(*data-value*)
> Specifies the 8-character name of the PIPELINE about which you are inquiring.

ENABLESTATUS(*cvda*)
> Specifies the status of the PIPELINE:

ENABLED
    Inbound service requests for this PIPELINE are processed normally.

DISABLED
    Inbound service requests for this PIPELINE are rejected.

SET WORKREQUEST command

There is a new value for the `WORKTYPE` option on the **SET WORKREQUEST** command:

SOAP
    Specifies that the work is being executed for a Web service request.

INQUIRE WEBSERVICE

Use the INQUIRE WEBSERVICE command to retrieve information about an installed WEBSERVICE.

INQUIRE WEBSERVICE

►►──INQUIRE WEBSERVICE(*name*)──┬─────────────────────────┬──►◄

    ├─BINDING(*data-area*)────┤
    ├─CONTAINER(*data-area*)──┤
    ├─ENDPOINT(*data-area*)───┤
    ├─LASTMODTIME(*data-area*)┤
    ├─PGMINTERFACE(*cvda*)────┤
    ├─PIPELINE(*data-area*)───┤
    ├─PROGRAM(*data-area*)────┤
    ├─STATE(*cvda*)───────────┤
    ├─URIMAP(*data-area*)─────┤
    ├─VALIDATIONST(*cvda*)────┤
    ├─WSBIND(*data-area*)─────┤
    └─WSDLFILE(*data-area*)───┘

Conditions: NOTAUTH, NOTFND
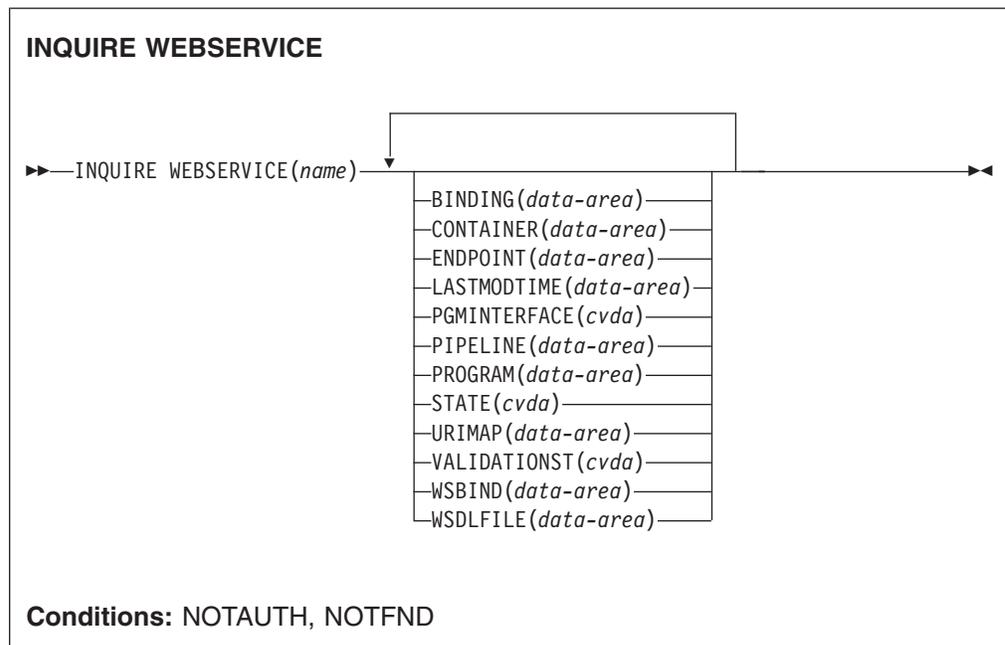
Browsing

You can browse through all the WEBSERVICEs installed in your system by using the browse options (START, NEXT, and END) on INQUIRE WEBSERVICE commands. See Browsing resource definitions for general information about browsing, including syntax, exception conditions, and examples.

Options

BINDING(*data-area*)
    Returns the WSDL binding represented by the WEBSERVICE. This binding is one of (potentially) many that appear in the WSDL file. The name can be up to 255 characters long.

CONTAINER(*data-area*)
    Returns the name of the container used if PGMINTERFACE returns a value of CHANNEL. The name can be up to 16 characters long.

ENDPOINT(*data-area*)
    Returns the endpoint URI of a remote WEBSERVICE. This is the endpoint URI

specified in the WSDL file for a remote Web service. If a CICS application program is the service provider, then the ENDPOINT will be empty. The URI can be up to 255 characters long.

LASTMODTIME(*data-area*)

Returns the time, in milliseconds since 00:00 on January 1st 1900, that the deployed WSBind file on HFS was last updated. This is a readonly value that CICS updates when the WEBSERVICE resource is installed or updated. The last-modified-time can be used to determine whether CICS has refreshed itself after an update is made to a WSBind file in the pickup directory.

- For dynamically-installed WEBSERVICEs (those installed by the CICS scanning mechanism), the value of LASTMODTIME is the timestamp of the HFS file pointed to by the WSBind definition, at the time the WEBSERVICE definition was last installed or updated.
- For statically-installed WEBSERVICEs (those installed from a CSD or by CREATE WEBSERVICE), the value of LASTMODTIME is the timestamp of the WSBind HFS file pointed to by the WEBSERVICE definition, at the time the WEBSERVICE was installed.

If you issue an INQUIRE WEBSERVICE command before a newly-installed or updated WEBSERVICE has fully initialized, the returned LASTMODTIME value will be zero.

The value is returned in 8-byte packed-decimal form. You can use the EXEC CICS FORMATTIME command to convert the LASTMODTIME value to the date-and-time format that you prefer.

PGMINTERFACE(*cvda*)

Returns a CVDA indicating whether the CICS program that implements the Web service expects input in a channel or in a commarea:

CHANNEL

The program expects input in a channel.

COMMAREA

The program expects input in a commarea

PIPELINE(*data-area*)

Returns the name of the PIPELINE in which the WEBSERVICE is installed; that is, the name of the PIPELINE resource that contains this WEBSERVICE resource. The name can be up to 8 characters long.

PROGRAM(*data-area*)

Returns the name of a CICS program that implements the Web service. If this WEBSERVICE represents a remote Web service (that is, CICS is not the service provider), PROGRAM will be empty. The name can be up to 8 characters long.

STATE(*cvda*)

Returns a CVDA indicating the state of the WEBSERVICE:

DISCARDING

A DISCARD command has been issued for the WEBSERVICE. The WEBSERVICE is quiescing before being discarded. It is not accepting new work, but is allowing currently-executing work to complete.

INITING

The Web service binding file, and the WSDL file, are being copied to the shelf.

INSERVICE

Resolution of the copy of the WSBIND file on the shelf has succeeded, and the WEBSERVICE is usable.

UNUSABLE

Copying of the WSBIND file on the shelf has failed, and the WEBSERVICE is unusable.

URIMAP(*data-area*)

Returns the name of a dynamically installed URIMAP if there is one that is associated with this WEBSERVICE. If the WEBSERVICE was not installed by performing the SCAN function on a PIPELINE resource, or if the WEBSERVICE represents a remote Web service, then the URIMAP will be empty. The name can be up to 8 characters long.

VALIDATIONST(*cvda*)

Returns a CVDA indicating whether full validation of SOAP messages is currently enabled for this WEBSERVICE:

VALIDATION

Full validation is enabled.

DISABLED

Full validation is disabled.

WEBSERVICE(*name*)

Specifies the name of the WEBSERVICE about which you are inquiring. The name can be up to 32 characters long.

WSBIND(*data-area*)

Returns the name of the Web service binding file. The name can be up to 255 characters long.

WSDLFILE(*data-area*)

Returns the name of the Web service description file associated with the WEBSERVICE resource. The name can be up to 255 characters long.

SET WEBSERVICE

Use the **SET WEBSERVICE** command to change the status of an installed WEBSERVICE.

SET WEBSERVICE

```
►►──SET WEBSERVICE(name)──────────────────────────────────────►◄
                         ─VALIDATIONST(cvda)─
                         ─VALIDATION─────────
                         ─NOVALIDATION───────
```

Conditions: INVREQ, NOTAUTH, NOTFND
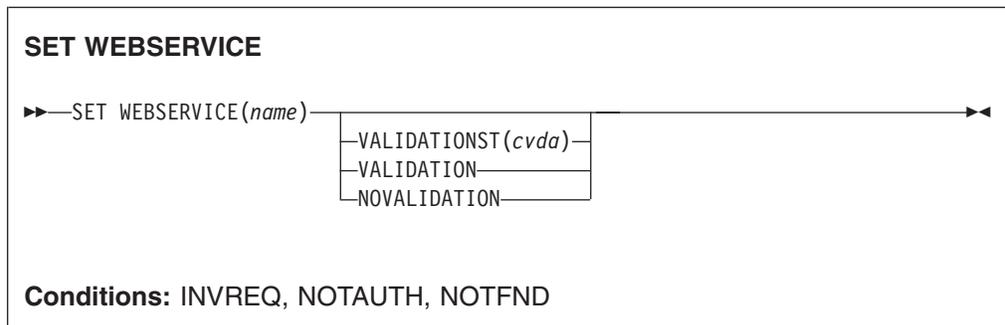
Options

WEBSERVICE(*name*)

Specifies the name of the WEBSERVICE.

VALIDATIONST(*cvda*)
> Specifies whether full validation is enabled for the WEBSERVICE or not. CVDA values are:

> **VALIDATION**
>> Full validation is enabled.

> **NOVALIDATION**
>> Full validation is not enabled.

Changes to CEMT

CEMT DISCARD WEBSERVICE

Use the **DISCARD WEBSERVICE** command to remove a WEBSERVICE from your CICS region.

CEMT DISCARD PIPELINE

Use the **DISCARD PIPELINE** to remove a PIPELINE from your CICS region.

CEMT INQUIRE WEBSERVICE

Use the **INQUIRE WEBSERVICE** command to display the following information about an installed WEBSERVICE:

```
>>──CEMT INQUIRE WEBSERVICE──┬──ALL──────┬──────────────────────────────────────>
                            └─(value)─┘  └─BINDING(value)─┘

>──┬─DATESTAMP(value)─┬──┬─DISCARDING─┬──┬─ENDPOINT(value)─┬──────────────────────>
                       ├─INITING────┤
                       ├─INSERVICE──┤
                       └─UNUSABLE───┘

>──┬─PIPELINE(value)─┬──┬─CHANNEL──┬──┬─PORTTYPE(value)─┬──────────────────────────>
                       └─COMMAREA─┘

>──┬─PROGRAM(value)─┬──┬─TIMESTAMP(value)─┬──┬─URIMAP(value)─┬─────────────────────>

>──┬─VALIDATION───┬──┬─WSBIND(value)─┬──┬─WSDLFILE(value)─┬───────────────────────><
   └─NOVALIDATION─┘
```
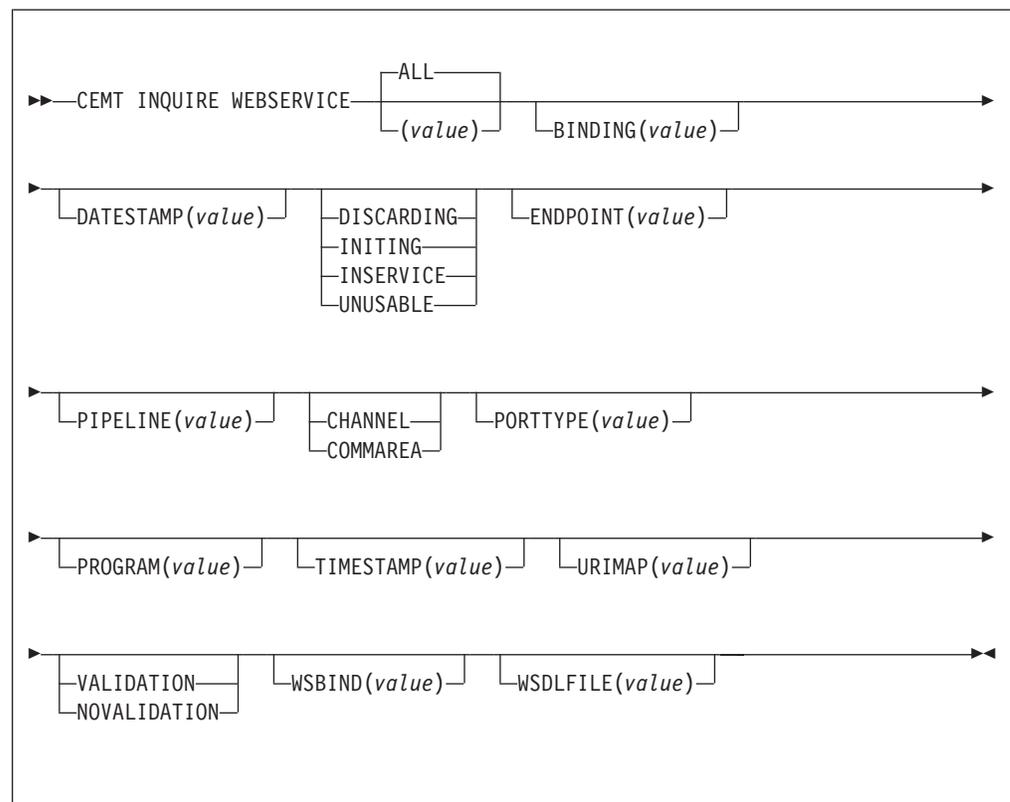
BINDING(*value*)
> Displays the WSDL binding represented by the WEBSERVICE. This binding is one of (potentially) many that appear in the WSDL file.

DATESTAMP(*value*)
> Displays the date, in yyyymmdd format, that the WEBSERVICE was last
> updated. This is a readonly value that CICS updates when the WEBSERVICE
> resource is installed or updated.

COMMAREA
> The program expects to receive input in a COMMAREA.

CONTAINER
> The program is expecting to receive input in a CONTAINER.

DISCARDING
> A DISCARD operation is in progress for this WEBSERVICE.

ENDPOINT(*value*)
> Displays the endpoint URI of a remote WEBSERVICE. This is the endpoint URI
> specified in the WSDL file for a remote Web service. If a CICS application
> program is the service provider, then the ENDPOINT will be empty.

INITING
> The WSBind file and WSDL file are being copied to the shelf.

INSERVICE
> Resolution of the copy of the WSBind file on the shelf has succeeded and the
> WEBSERVICE is usable.

NOVALIDATION
> Full validation is not enabled.

PIPELINE(*value*)
> Displays the name of the PIPELINE in which the WEBSERVICE is installed;
> that is, the name of the PIPELINE resource that contains this WEBSERVICE
> resource.

PORTTYPE(*value*)
> Displays the contents of the <portType> element within the WSDL document
> represented by this WEBSERVICE resource.

PROGRAM(*value*)
> Displays the name of the CICS program that implements the WSDL <portType>
> when CICS is the service provider.

TIMESTAMP(*value*)
> Displays the time, in hh:mm:ss format, that the WEBSERVICE was last
> updated. This is a readonly value that CICS updates when the WEBSERVICE
> resource is installed or updated.

URIMAP(*value*)
> Displays the name of a dynamically installed URIMAP if there is one that is
> associated with this WEBSERVICE. If the WEBSERVICE was not installed by
> performing the SCAN function on a PIPELINE resource, or if the WEBSERVICE
> represents a remote Web service, then the URIMAP will be empty.

UNUSABLE
> Copying the WSBind file to the shelf has failed and the WEBSERVICE is
> unusable.

VALIDATION
> Full validation is enabled.

WEBSERVICE(**ALL**|*value*)

ALL
    is the default. Information about all WEBSERVICEs is displayed, unless you
    specify a selection of WEBSERVICEs to be queried.

value
    is the name (1-8 characters) of an installed WEBSERVICE definition.

WSBIND(*value*)
    Returns the 1-255 character fully-qualified file name of the WSBind file on HFS.

WSDLFILE(*value*)
    Returns the 1-255 character fully-qualified file name of the WSDL file on HFS.

CEMT INQUIRE WORKREQUEST

There is a new value for the WORKTYPE option on the **INQUIRE WORKREQUEST**
command:

PIPE
    Specifies that the work is being executed for a Web service request.

CEMT INQUIRE PIPELINE

Use the **INQUIRE PIPELINE** command to display information about an installed
PIPELINE.
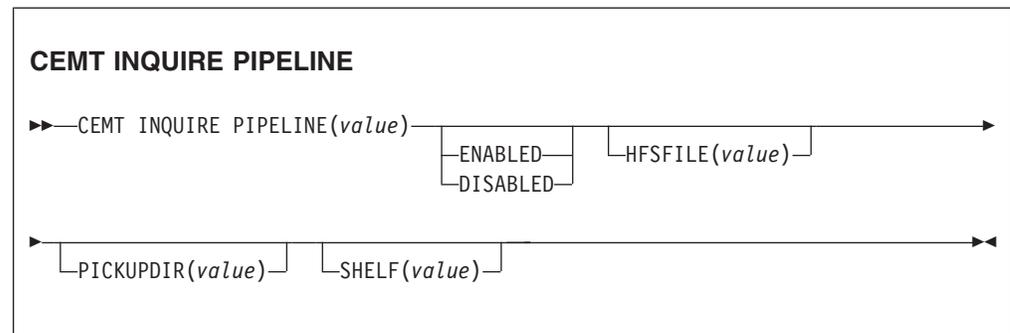
CEMT INQUIRE PIPELINE

▶▶──CEMT INQUIRE PIPELINE(*value*)──▶
                                    └─ENABLED──┘ └─HFSFILE(*value*)─┘
                                    └─DISABLED─┘

▶──▶◀
   └─PICKUPDIR(*value*)─┘ └─SHELF(*value*)─┘

PIPELINE(*value*)
    Specifies the 8-character name of the PIPELINE about which you are inquiring.

HFSFILE(*value*)
    Returns the name of the HFS file that contains information about the processing
    nodes that will act on a service request, and on the response.

ENABLED
    Inbound service requests for this PIPELINE are processed normally.

DISABLED
    Inbound service requests for this PIPELINE are rejected.

PICKUPDIR(*value*)
    Returns the 1–255 character fully-qualified name of the *pickup directory* on
    HFS.

SHELF(*value*)
    Returns the 1–255 character fully-qualified name of a *shelf directory* on HFS.

CEMT PERFORM STATISTICS

This command supports the following new options:

PIPELINE
> Statistics related to a PIPELINE are written immediately to the SMF data set. The statistics include information about HFS files.

WEBSERVICE
> Statistics related to a WEBSERVICE are written immediately to the SMF data set.

CEMT PERFORM PIPELINE

Use the **CEMT PERFORM PIPELINE** command to initiate a scan of the directory named in the WSDIR attribute of a PIPELINE.

CEMT SET WEBSERVICE

Use the **SET WEBSERVICE** command to change the status of an installed WEBSERVICE.

```
                              ┌─ALL─┐
►►──CEMT SET WEBSERVICE────────┼─────┼──┬──────────────┬──────────►◄
                              └─(value)─┘  ├─VALIDATION───┤
                                          └─NOVALIDATION─┘
```
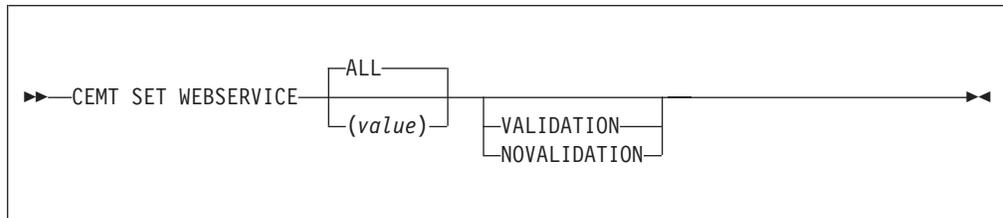
WEBSERVICE(ALL|*value*)

> **ALL**
> > Any changes you request are made to all resources of the specified type that you are authorized to access.
>
> *value*
> > is the name (1-8 characters) of an installed WEBSERVICE definition.

Validation
> Full validation is enabled

Novalidation
> Full validation is not enabled

CEMT SET WORKREQUEST command

There is a new value for the WORKTYPE option on the **SET WORKREQUEST** command:

PIPE
> Specifies that the work is being executed for a Web service request.

CEMT SET PIPELINE

Use the **SET PIPELINE** command to change the status of an installed PIPELINE.

```
 ┌──────────────────────────────────────────────────────────────────┐
 │  CEMT SET PIPELINE                                                 │
 │                                                                    │
 │                              ┌─ALL─┐                               │
 │  ►►──CEMT SET PIPELINE───────┼─────┼──┬──────────┬────────────►◄   │
 │                              └─(value)─┘ ├─ENABLED──┤               │
 │                                          └─DISABLED─┘               │
 │                                                                    │
 └──────────────────────────────────────────────────────────────────┘
```
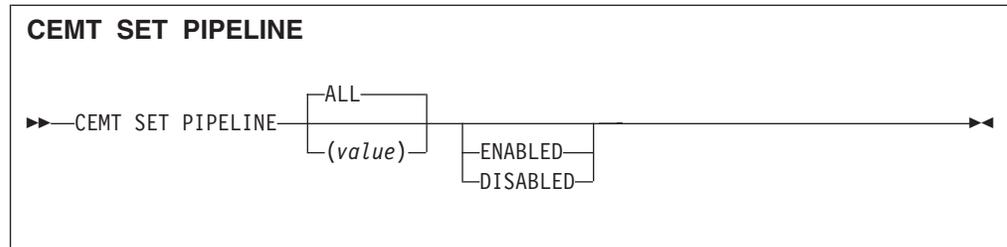
PIPELINE(*value*)
> Specifies the 8-character name of the PIPELINE.

ENABLED
> Specifies that inbound service requests for this PIPELINE are to be processed normally.

DISABLED
> Specifies that inbound service requests for this PIPELINE are to be rejected.

Changes to the JCICS API

There are JCICS equivalents of the following commands:

> **EXEC CICS FAULT ADD**
>
> **EXEC CICS FAULT CREATE**
>
> **EXEC CICS FAULT DELETE**
>
> **EXEC CICS INVOKE WEBSERVICE**

Changes to CICS-supplied transactions

Note: Changes to CEMT are described in "Changes to CEMT" on page 71.

Changes to CETR

Transaction CETR now supports the PI domain.

Changes to statistics

The following new DSECTs will be provided:

DFHPIPDS
> Provides statistics for the PIPELINE resource.

DFHPIWDS
> Provides statistics for the WEBSERVICE resource

Changes to sample programs

The following programs now support the PIPELINE and WEBSERVICE resources:

> DFH0STAT, the sample statistics program
>
> DFH$FORA, the DB2® formatting sample program (assembler)
>
> DFH$FORP, the DB2 formatting sample program (PL/I)
>
> DFH0FORC, the DB2 formatting sample program (COBOL)
>
> DFH$DB2T, the DB2 table definitions for DFH$FORA, DFH$FORP, and DFH0FORC
>
> DFH$SQLT, input for the DB2 table load utility

Changes to CICS utilities

Statistics utility program DFHSTUP

DFHSTUP supports the changes to statistics described in "Changes to statistics" on page 75.

The resource types that you can code on the `SELECT TYPE` and `IGNORE TYPE` control parameters for DFHSTUP now include:
- WEBSERVICE
- PIPELINE

Changes to problem determination

The following domains are new in CICS Transaction Server for z/OS, Version 3 Release 1:

Component ID	Description
PI	Pipeline manager domain

To support the new domains:
- CICS trace functions (including CETR, trace-related system initialization parameters, and the trace utility program) now support the component IDs listed.
- CICS system dumps include control blocks for the new domains. The CICS-supplied dump exit routine supports the use of the component IDs listed, to specify the control blocks to be included in the dump output.
- CICS messages issued by the domains listed above use a message prefix of the form `DFHcomponent-ID`.

Security

Security for new SPI and CEMT commands

New predefined RACF® resource names will control access to the following resources using the SPI and CEMT:

    PIPELINE

    WEBSERVICE

New Category 1 transactions

The following new transaction is for CICS internal use, and should not be invoked from a user terminal. For security purposes, it is a category 1 transaction.

    CPIS

Migration and coexistence

This topic describes how Web services in CICS affects migration to CICS Transaction Server for z/OS, Version 3, and how it coexists with other functions.

Migration of existing functions

If you use the SOAP for CICS feature, you must perform a number of tasks to migrate applications that use the feature. The support for Web services provided in CICS Transaction Server for z/OS, Version 3 Release 1 is substantially different from that provided in the feature.

The SOAP for CICS feature relies to a considerable extent upon user-written code, and therefore it is not possible to set out a step-by-step migration task. However, here are some of the things you will need to think about.

- Consider using the Web services assistant to construct and parse SOAP messages. If you decide to do so, you are advised to discard your existing message adapters, and deign new wrapper programs to replace them, as it is unlikely that you will be able to reuse significant amounts of code in your adapters.
- If you use SOAP messages, but decide not to use the Web services assistant, you may be able to reuse your existing code for constructing and parsing the messages. However, you should consider whether to use the CICS-provided SOAP message handlers, because they are designed to work with SOAP 1.1 and SOAP 1.2 messages.
- Review your use of containers. The SOAP for CICS feature uses BTS containers, whereas CICS Transaction Server for z/OS, Version 3 Release 1 uses channel containers. You will need to review your programs and change any BTS-related commands required by the feature. You will also need to review the name and usage of each container, as most of these have changed.
- Consider how to migrate the function that was provided by your pipeline programs. The pipeline in the SOAP for CICS feature has a fixed number of user-written programs, each with a designated purpose. The function provided by some of these programs is provided in CICS Transaction Server for z/OS, Version 3 Release 1 by the CICS-provided SOAP message handlers, so you may be able to dispense with these programs altogether.

  On the other hand, CICS Transaction Server for z/OS, Version 3 Release 1 lets you define as many programs in your pipeline as you need. Therefore, you should consider whether the function performed by your pipeline programs should be restructured to take advantage of the new framework.

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

  In the SOAP for CICS feature, you could have just one pipeline for all your service provider applications, and one for all your service requesters. In CICS Transaction Server for z/OS, Version 3 Release 1, you can configure many different pipelines. Therefore, it is possible that the logic you provided in your pipeline programs to distinguish one application from another can be replaced by CICS resource definitions. For example, in a service provider, code that distinguishes between applications based upon a URI, can be replaced with a suitable set of URIMAP resources

Coexistence

If you use the SOAP for CICS feature, you can continue to do so; the feature continues to be fully supported in CICS Transaction Server for z/OS, Version 3 Release 1, independently of Web services in CICS.

The SOAP for CICS feature can interoperate with the support for Web services in CICS TS for z/OS, Version 3.1: the feature can be the service requester or the service provider.

CICSPlex SM support

There are changes to CICSPlex SM views and resource tables to support Web services in CICS.

Changes to the CICSPlex SM application programming interface

The following new resource tables have been introduced:
- "PIPEDEF resource table"
- "PIPELINE resource table"
- "WEBSVDEF resource table" on page 79
- "WEBSERV resource table" on page 79

PIPEDEF resource table

The PIPEDEF resource table includes the following RDO attributes:

PIPELINE
> Pipeline definition name

STATUS
> Shows the CVDA value that indicates if the PIPELINE is to be installed in ENABLED (default) or DISABLED state

CONFIGFILE
> The 255-character fully-qualified configuration file name on HFS for this Pipeline

SHELF
> The 255-character fully-qualified name of the WSBind directory (shelf)

WSDIR
> The 255-character name of the WSBind (pickup) directory on HFS

PIPELINE resource table

The PIPELINE resource table has the following SPI attributes:

PIPELINE
> Pipeline definition name

ENABLESTATUS
> The CVDA value that specifies the ENABLE status of the Pipeline

PIPEUSECOUNT
> Pipeline use count

CONFIGFILE
> The 255-character fully-qualified configuration file name on HFS for this Pipeline

SHELF
> The 255-character name of a directory (shelf) for WSBind files

WSDIR
> The 255-character name of the WSBind (pickup) directory on HFS

WEBSVDEF resource table

The WEBSVDEF resource table includes the following RDO attributes:

WEBSERVICE
WEBSERVICE definition name

PIPELINE
The Pipeline in which this Web service is to be installed

WSBIND
The 255-character fully-qualified name of the WSBind file on HFS

WSDLFILE
The 255-character fully-qualified name of the WSDL file on HFS

VALIDATION
Shows the CVDA value (YES or NO) to indicate whether or not a full
validation of WSDL SOAP messages will be performed

WEBSERV resource table

The WEBSERV resource table includes the following SPI attributes:

WEBSERVICE
WEBSERVICE name

PIPELINE
The Pipeline in which this Web service is installed

WSBIND
The 255-character fully-qualified name of the WSBind file on HFS

WSDLFILE
The 255-character fully-qualified name of the WSDL file on HFS

URIMAP
The dynamically-installed URIMAP associated with this Web service

BINDING
The WSDL Binding that this Web service represents

ENDPOINT
The ENDPOINT URI of a remote Web service

PROGRAM
The 8-character CICS application program name that implements this Web
Service

PGMINTERFACE
The CVDA value (CHANNEL or COMMAREA) indicating where the
specified PROGRAM expects input

CONTAINER
The name of the container used if the PGMINTERFACE is CHANNEL

VALIDATIONST
The CVDA value (VALIDATION or NOVALIDATION) indicating whether or
not the full validation of WSDL SOAP messages should be performed

LASTMODTIME
The time that the deployed WSBind file on HFS was last updated

STATE

The CVDA value (DISCARDING, INITING, INSERVICE, UPDATING, or UNUSABLE) indicating the state of the Web service

WEBUSECOUNT

Web service use count

Changed resource table

The following resource table has been changed:
*   "WORKREQ"

WORKREQ

The WORKREQ resource table now includes the following attribute:

WORKTYPE

Type of work being performed. CVDA IIOP and SOAP types are the only types available.

Changes to CICSPlex SM Web User Interface

New WUI views

The following WUI views have been introduced:
*   "Pipeline definitions view"
*   "Pipeline view" on page 81
*   "Web service definition view" on page 81
*   "Web service view" on page 81

Pipeline definitions view

A new definitional view set has been introduced called **Pipeline definitions**, associated with the new PIPEDEF resource table. The view name for this tabular view is EYUSTARTPIPEDEF.TABULAR. and the existing EYUSTARTADMRES menu has been extended to include it.

To open the **Pipeline definitions** view, do the following:
1. Click **Administration views** from the Main menu
2. Click **Basic CICS resource administration views** (or, alternatively, click **Fully functional Business Application Services (BAS) administration views**)
3. Click **CICS resource definitions**
4. Scroll down and click **Pipeline definitions**

The **Pipeline definitions** view is displayed. This view includes the following five action buttons:
*   **Create**
*   **Update**
*   **Remove**
*   **Install**
*   **Add to Resource group**

See the attributes of the PIPEDEF resource table listed in "PIPEDEF resource table" on page 78 for field details.

Pipeline view

A new view has been introduced called **Pipeline**, associated with the new PIPELINE resource table. The view name for this tabular view is EYUSTARTPIPELINE.TABULAR and the existing EYUSTARTTCPIPS menu has been extended to include it.

To open the **Pipeline** view, do the following:
1. Click **CICS operations** view from the Main menu
2. Scroll down and click **TCP/IP service operations views**
3. Click **Pipeline**

The **Pipeline** view is displayed.

See the attribute details of the PIPELINE resource table listed in "PIPELINE resource table" on page 78 for field details.

Web service definition view

A new definitional view set has been introduced called **Web service definition**, associated with the new WEBSVDEF resource table. The view name for this tabular view is EYUSTARTWEBSVDEF.TABULAR. and the existing EYUSTARTADMRES menu has been extended to include it.

To open the **Web service definition** view, do the following:
1. Click **Administration views** from the Main menu
2. Click **Basic CICS resource administration views** (or, alternatively, click **Fully functional Business Application Services (BAS) administration views**)
3. Click **CICS resource definitions**
4. Scroll down and click **Web service definitions**

The **Web service definition** view is displayed This view includes the following five action buttons:
- **Create**
- **Update**
- **Remove**
- **Install**
- **Add to Resource group**

See the attributes of the WEBSVDEF resource table listed in "WEBSVDEF resource table" on page 79 for field details.

Web service view

A new tabular view has been introduced called **Web service**, associated with the new WEBSERV resource table. The view name for this tabular view is EYUSTARTWEBSERV.TABULAR and the existing EYUSTARTTCPIPS menu has been extended to include it.

To open the **Web service** view, do the following:
1. Click **CICS operations** view from the Main menu
2. Scroll down and click **TCP/IP service operations views**
3. Click **Web service**

The **Web service** view is displayed.

See the attributes of the WEBSERV resource table listed in "WEBSERV resource table" on page 79 for field details.

Changed WUI view

The following WUI view has been changed:

- "Work Request view"

Work Request view

The following attribute has been added to the **Work Request** view within the Task operations view:

WORKTYPE
> The type of work being performed. IIOP and SOAP are the only types available.

Chapter 3. Support for HTTP client requests from CICS applications

You can now create outbound requests from a CICS application through CICS as an HTTP client, using **EXEC CICS** commands.

The facility for CICS to act as an HTTP client is fully integrated into CICS Web support. You can use **EXEC CICS** commands in CICS application programs to open an HTTP connection to a server, make requests, and receive responses for processing by the application program. Some new **EXEC CICS** commands are introduced for this purpose, and some existing commands have been given outbound as well as inbound options.

Benefits of support for HTTP client requests from CICS applications

The **EXEC CICS** commands for CICS as an HTTP client mean that CICS application programs can be written to:

- Use a common protocol for business-to-business communications.
- Control hardware or software using the HTTP protocol (for example, printers can sometimes be controlled in this way).
- Access HTTP applications that provide items of information (for example, share prices) and retrieve this information for use in the application.

Global user exits in CICS Web support processes enable you to specify the use of proxy servers, and to apply a security policy, for outbound HTTP requests from CICS.

Requirements

There are no special hardware or software requirements to support this function.

Related information:

Chapter 27, "The CICS operating environment," on page 355

HTTP request and response processing for CICS as an HTTP client

For CICS as an HTTP client, CICS is the Web client, and it communicates with an HTTP server. A user-written application program sends requests through CICS to the HTTP server, and receives the responses from it. CICS maintains a persistent connection with the server. A session token is used on the commands issued by the application program to identify the connection.

An application program that makes an HTTP request and receives a response must use the **EXEC CICS** WEB API commands to explicitly direct the interaction with the server.

The application program that initiates the HTTP request should be designed to process whatever CICS receives from the server in response to that request, which might include error responses, redirection to another URL, embedded hypertext links, HTML forms, image source, or other items that request an action from the application program. CICS can perform code page conversion for requests and responses, if required.

83

This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

Processing for CICS as an HTTP client takes place as follows:

1. **The application program initiates a connection with the HTTP server through CICS.** The application program does this by issuing the `EXEC CICS` WEB OPEN command. A URIMAP resource definition that you have created can be referenced to specify the scheme and host name for the connection, or the application program can provide this information.

2. **CICS establishes the connection with the server.** Using the information provided by the application program, CICS opens a TCP/IP connection on a socket and contacts the server. When the TCP/IP connection is established, CICS returns a session token to the application program to uniquely identify the connection. This session token is used on all the remaining commands issued by the application program concerning that connection.

3. **The application program may write HTTP headers for its request.** User-written HTTP headers can be built using the WEB WRITE HTTPHEADER command and stored ready for sending.

4. **The application program specifies components of the request line.** The request method, path information and query string are specified using the WEB SEND or WEB CONVERSE command.

5. **The application program may produce an entity body for its request.** The content of the request is specified on the WEB SEND or WEB CONVERSE command.

6. **The application program initiates transmission of the request.** When the application program issues the WEB SEND or WEB CONVERSE command, the request is passed to CICS for sending across the connection specified by the session token.

7. **CICS generates some required HTTP headers and adds them to the request, then sends the request to the server.**

8. **The server receives and processes the request, and provides a response.** CICS passes the response to the application program.

9. **The application program examines the response.** The WEB READ HTTPHEADER command, or the HTTP header browsing commands, can be used to examine the headers of the response. The WEB RECEIVE or WEB CONVERSE command receives the body of the response (if there is one), which can be processed by the application program, and the response's status code and status text.

10. **The application program may initiate further requests and responses.** If the server supports persistent connections, the connection identified by the session token remains open for further requests.

11. **The application program initiates closing of the connection to the server.** When all the requests and responses are completed, the application program issues a WEB CLOSE command, and CICS closes its end of the TCP/IP connection. If the application program does not issue a WEB CLOSE command, the connection is closed at end of task.

During this process, code page conversion is usually needed when messages enter and leave the CICS environment, so that CICS Web support processing and user-written applications (which typically use an EBCDIC encoding) can communicate with HTTP servers (which typically use an ASCII encoding).

Session tokens

A session token is an 8-byte binary value that uniquely identifies a connection between CICS as an HTTP client, and an HTTP server. The use of a session token for each connection means that CICS Web support can manage multiple connections to servers by different tasks, and also means that an application program can control more than one connection.

A connection begins in response to a WEB OPEN command issued by a user application program. The session token is returned on successful completion of the WEB OPEN command, and used on all the `EXEC CICS` WEB commands issued by the application program concerning that connection.

Using the connection, the user application program can make HTTP client requests to the server, and receive responses from it. The connection can persist for more than one exchange of a request and a response, until either the application program or the server chooses to terminate the connection. "How CICS Web support handles persistent connections" on page 115 has more detail about how CICS Web support handles persistent connections and how they are terminated.

If the server terminates the connection, the application program cannot send any further requests using that connection, but it can read the response that the server sent before it terminated the connection. The session token remains valid for use on commands to access that data, until the application issues the WEB CLOSE command. After the WEB CLOSE command is issued, the session token that applies to the connection is no longer valid. If the application program does not issue a WEB CLOSE command, the connection is closed at end of task.

The maximum number of open client connections, each represented by a session token, that can be present simultaneously in a CICS region is 32768.

Changes to CICS externals

Changes to resource definition

URIMAP definitions can be used as a convenient way to make an HTTP request through CICS as an HTTP client.

URIMAP definitions for CICS as an HTTP client

URIMAP definitions may be used to avoid specifying information such as a URL or a certificate label in an application program that makes an HTTP client request. The attributes of a URIMAP definition are listed in "Changes to resource definition" on page 135. A URIMAP definition for CICS as an HTTP client is defined with USAGE(CLIENT), and the SCHEME, HOST, PATH, CERTIFICATE and CIPHERS attributes can be specified. An application program can name the URIMAP definition on the WEB OPEN and WEB SEND commands to use the relevant information from it.

URIMAP definitions are also used for CICS as an HTTP server, and for Web services.

Changes to the monitoring control table, DFHMCT

The new monitoring fields 331-338 in the DFHWEBB group of performance class records are added to the monitoring control table. These fields monitor actions for CICS as an HTTP client.

Changes to the application programming interface (HTTP client requests)

New and changed commands

The following new `EXEC CICS WEB` commands are provided for CICS as an HTTP client:

- `EXEC CICS WEB OPEN`
- `EXEC CICS WEB CONVERSE`
- `EXEC CICS WEB CLOSE`

The following `EXEC CICS WEB` commands have a new range of options when used for CICS as an HTTP client:

- `EXEC CICS WEB SEND`
- `EXEC CICS WEB RECEIVE`

The following new `EXEC CICS` commands can be used for both CICS as an HTTP client, and CICS as an HTTP server, and are described in "Changes to the application programming interface (General CICS Web support enhancements)" on page 146:

- `EXEC CICS WEB PARSE URL`
- `EXEC CICS CONVERTTIME`

There are changes to the options available on most of the remaining `EXEC CICS WEB` commands. The changed commands can be used for both CICS as an HTTP client, and CICS as an HTTP server. The changes to the following commands are described in this section:

- `EXEC CICS WEB WRITE HTTPHEADER`
- `EXEC CICS WEB READ HTTPHEADER`
- `EXEC CICS WEB STARTBROWSE HTTPHEADER`
- `EXEC CICS WEB READNEXT HTTPHEADER`
- `EXEC CICS WEB ENDBROWSE HTTPHEADER`

The changes to the following commands are described in "Changes to the application programming interface (General CICS Web support enhancements)" on page 146:

- `EXEC CICS WEB EXTRACT`
- `EXEC CICS FORMATTIME`

The WEB FORMFIELD commands and the WEB RETRIEVE command cannot be used for CICS as an HTTP client.

WEB OPEN

Open a connection to a server for CICS as an HTTP client.

WEB OPEN

```
►►──WEB──OPEN──────────────────────────────────────────────────────────────────►

 ►──┬─URIMAP(data-value)────────────────────────────────────────────────┬──────►
    └─HOST(data-value)──HOSTLENGTH(data-value)──PORTNUMBER(data-value)──SCHEME(cvda)─┘

 ►──┬──────────────────────────┬──┬──────────────────────────────────────┬──────►
    └─CERTIFICATE(data-value)──┘  └─CIPHERS(data-value)──NUMCIPHERS(data-value)─┘

 ►──┬─────────────────────┬──SESSTOKEN(data-area)──HTTPVNUM(data-area)──HTTPRNUM(data-area)──►◄
    └─CODEPAGE(data-value)─┘
```

Conditions: IOERR, INVREQ, LENGERR, NOTFND, NOTAUTH, TIMEDOUT

This command is threadsafe.
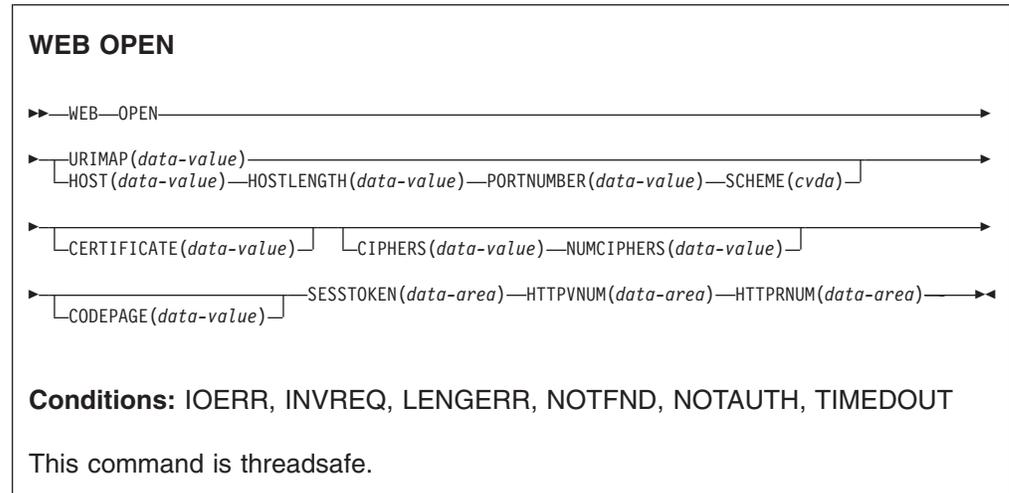
Description

WEB OPEN enables an application program, through CICS Web support, to open a connection with a specified host on an HTTP server on the Internet. The host name and scheme can be used from a preset URIMAP definition, which also supplies a default path for requests.

When the connection is open, the application program can make HTTP client requests to the server and receive responses from it. CICS queries the HTTP version of the server (using an OPTIONS request) when the connection is opened, and uses this information for subsequent communications. CICS also returns the HTTP version information to the application program, to be checked if you plan to write HTTP headers or send chunked information.

The WEB OPEN command drives the XWBOPEN user exit, which can make the connection to the server go through a proxy server, if required.

Options

CERTIFICATE(*data-value*)
specifies the label of the X.509 certificate that is to be used as the SSL client certificate during the SSL handshake. Certificate labels can consist of up to 32 alphanumeric characters. This option is only relevant when SCHEME(HTTPS) is specified. If SCHEME(HTTPS) is specified, but the CERTIFICATE option is omitted, the default certificate defined in the key ring for the CICS region user ID is used. The certificate must be stored in a key ring in the external security manager's database.

CIPHERS(*data-value*)
specifies a string of up to 56 hexadecimal digits that is interpreted as a list of up to 28 2-digit cipher suite codes. The cipher suite codes are used when SSL is active for the connection, so this option is only relevant when SCHEME(HTTPS) is specified. They indicate the method of encryption to be used for this connection.

Use the NUMCIPHERS option to specify the number of cipher suite codes in your list. The codes that are available depend on what level of encryption has been specified by the ENCRYPTION system initialization parameter. If you specify any cipher codes that are not in the default list for the active encryption level, they are ignored.

You can specify the URIMAP option to use this information directly from an existing URIMAP definition, in which case the CIPHERS option is not required. You may still specify the CIPHERS option, and the setting in the URIMAP definition is overridden by any codes that you specify for this option.

If you omit the CIPHERS option and the URIMAP option, but SSL is active for
the connection, the default cipher list for the encryption level for the running
system is used.

CODEPAGE(*data-value*)
specifies a code page, usually EBCDIC, that is suitable for the application program. The code page name can be up to 8 alphanumeric characters. The default is the default code page for the local CICS region, as specified in the LOCALCCSID system initialization parameter. The code page applies for the duration of this connection. When the server returns a response to an HTTP request, if conversion is requested (which is the default), CICS converts the request body into this code page before passing it to the application.

HOST(*data-value*)
specifies the host name on the server to which you want to connect.

An IPv4 address can be used as a host name, but IPv6 addresses are **not** supported.

If a port number is required, do not include this with the host name, but use the PORTNUMBER option to specify it.

HOSTLENGTH(*data-value*)
specifies, as a fullword binary value, the length of the host name.

HTTPRNUM(*data-area*)
returns the release number for the HTTP version of the server, as a halfword binary value. (HTTPVNUM returns the version number.) For example, if the server is at HTTP/1.0 level, HTTPRNUM returns 0.

HTTPVNUM(*data-area*)
returns the version number for the HTTP version of the server, as a halfword binary value. (HTTPRNUM returns the release number.) For example, if the server is at HTTP/1.0 level, HTTPVNUM returns 1.

CICS obtains the HTTP version information when it opens the connection to the server. If the server does not provide HTTP version information, CICS assumes that it is at HTTP/1.0 level.

If your application program writes HTTP headers that might be unsuitable for a server at HTTP/1.0 level, or if you intend to send a chunked message to the server (which cannot be received by a server at HTTP/1.0 level), your application program should also consult the HTTP version information.

Note: CICS does not make any special provision for a server that is below HTTP/1.0 level. CICS behaves as though these servers were at HTTP/1.0 level, and returns HTTP/1.0 as the HTTP version.

NUMCIPHERS(*data-value*)
specifies, as a halfword binary value, the number of cipher suite codes that you specified for the CIPHERS option.

PORTNUMBER*(data-value)*
specifies the port number, as a fullword binary value. You only need to specify the port number if it is **not** the default for the specified scheme. For HTTP, the default port number is 80, and for HTTPS, the default port number is 443.

SCHEME*(cvda)*
specifies the scheme that is to be used for the connection to the server, which can be with or without SSL. CVDA values are:

HTTP   is the HTTP protocol, without SSL.

HTTPS
is the HTTPS protocol, which is HTTP with SSL. If HTTPS is used, the CICS address space must be enabled for SSL.

SESSTOKEN*(data-area)*
returns the session token, an 8-byte binary value that uniquely identifies this connection between CICS and a server. It is returned when the connection is opened successfully. The session token must be used on all CICS WEB commands that relate to this connection. "Session tokens" in the *CICS Internet Guide* explains the use of the session token.

URIMAP*(data-value)*
specifies the name (up to 8 characters, in mixed case) of a URIMAP definition that provides the following information:

- The scheme that is to be used for the connection to the server.
- The host name on the server to which you want to connect.
- A port number, if required.
- A path component for the URI, representing the resource on the server that you want to access. This path becomes the default path for WEB SEND or WEB CONVERSE commands relating to this connection, but it can be overridden by specifying another path on the WEB SEND or WEB CONVERSE command.
- The label of the X.509 certificate that is to be used as the SSL client certificate, if required.
- The cipher suite codes that can be used for the connection.

If the URIMAP option is specified, do not specify the CERTIFICATE, HOST, HOSTLENGTH, PORTNUMBER, PORTLENGTH, or SCHEME options. The CIPHERS and NUMCIPHERS options can be omitted or specified in the command; if specified, they override these settings in the URIMAP definition. The URIMAP definition must be for CICS as an HTTP client, with USAGE(CLIENT) specified.

WEB CLOSE

Closes a connection to a server for CICS as an HTTP client.

WEB CLOSE

►►──WEB──CLOSE──SESSTOKEN(*data-value*)──────────────────────────────►◄

Conditions: NOTOPEN

This command is threadsafe.

Description

WEB CLOSE enables an application program to close a connection with a server. The session token identifies the connection that is to be closed. When the connection is closed, the session token that applies to it is no longer valid for use. The session token is required to receive a response from the server and to read the HTTP headers for the response, so the WEB CLOSE command should not be issued until all interaction with the server and with the response that it sends is complete. The command releases CICS resources involved with the connection.

The WEB CLOSE command does not cause CICS to notify the server that the connection should be terminated. It only makes CICS close the connection on the client side. On the final request that you make using the connection, you should specify the CLOSESTATUS(CLOSE) option on the WEB SEND or WEB CONVERSE command. When this option is specified, CICS writes a Connection: close header on the request, or, for a server at HTTP/1.0 level, omits the Connection: Keep-Alive header. The information in the headers means that the server can close its connection with you immediately after sending the final response, rather than waiting to see if you send further requests before timing out.

The connection might also be closed at the request of the server before the WEB CLOSE command is issued. If you need to test whether the server has requested termination of the connection, use the WEB READ HTTPHEADER command to look for the Connection: close header in the last message from the server.

If the server does request termination of the connection, the data relating to that connection is still kept available within CICS until the WEB CLOSE command is issued. The available data includes the most recent message received from the server, and the parameters used to open the connection (such as the scheme and the host name of the server). When a server has terminated the connection, the application program cannot:
* Send further requests on that connection, using the WEB SEND or WEB CONVERSE commands.
* Write HTTP headers, using the WEB WRITE HTTPHEADER command.

However, the application program can still:
* Receive a response, using the WEB RECEIVE command.
* Examine HTTP headers, using the WEB READ HTTPHEADER and HTTP header browsing commands.
* Extract connection information, using the WEB EXTRACT command.

When the WEB CLOSE command is issued, the data relating to the connection is cleared.

If the WEB CLOSE command is not issued by the application program, then at end of task CICS clears the data relating to the connection and closes the connection, if it has not already been closed.

Options

SESSTOKEN(*data-value*)
> specifies the session token, an 8-byte binary value that uniquely identifies a connection between CICS and a server. This value is returned by a WEB OPEN command for CICS as an HTTP client. When you issue the WEB CLOSE command for the connection identified by the session token, CICS ends that connection and clears the data associated with it, and makes the session token invalid for further use by the application program. "Session tokens" in the *CICS Internet Guide* explains the use of the session token.

WEB SEND (Client)

Send an HTTP request by CICS as an HTTP client, using CICS Web support.

WEB SEND (CICS as an HTTP client)

```
►►─WEB──SEND──SESSTOKEN(data-value)──METHOD(cvda)─┬──────────────────────────────────────────┬─►
                                                  ├─PATH(data-area)──PATHLENGTH(data-value)──┤
                                                  └─URIMAP(data-value)───────────────────────┘

►─┬──────────────────────────────────────────┬───────────────────────────────────────────────►
  └─QUERYSTRING(data-area)──QUERYSTRLEN(data-value)─┘

►─┬──────────────────────────────────────────────────────────────────────┬───────────────────►
  └─MEDIATYPE(data-value)─┬───────────────────────────────────────────────┤
                          ├─DOCTOKEN(data-value)──────────────────────────┤
                          └─FROM(data-area)──FROMLENGTH(data-value)─┬──────────────────┘
                                                                    └─CHUNKING(cvda)─┘

►─┬─────────────────┬──┬────────────────────────┬──┬───────────────┬──┬──────────────────┬──►◄
  └─CLIENTCONV(cvda)─┘  └─CHARACTERSET(data-value)─┘  └─ACTION(cvda)─┘  └─CLOSESTATUS(cvda)─┘
```

Conditions: IOERR, INVREQ, LENGERR, NOTFND, NOTAUTH, NOTOPEN, TOKENERR

This command is threadsafe.

Description

WEB SEND for CICS as an HTTP client is used to make an HTTP request to a server. A session token must be included on this command.

Tip: For CICS as an HTTP client, the CONVERSE command can be used as an alternative to issuing a WEB SEND command followed by a WEB RECEIVE command. However, bear in mind that the WEB CONVERSE command does not support chunked transfer-coding, because this requires a sequence of send actions, and the WEB CONVERSE command provides a single send action.

Options

ACTION*(cvda)*

This option is used to specify how the message should be sent out. The CVDA value that applies for CICS as an HTTP client is:

EXPECT

makes CICS send an Expect header along with the request line and headers for the request, and await a 100-Continue response before sending the message body to the server. If a response other than 100-Continue is received, CICS informs the application program and cancels the send. If no response is received after a period of waiting, CICS sends the message body anyway.

This option must only be used if your request has a message body.

CHARACTERSET*(data-value)*

specifies the character set into which CICS translates the entity body of the request before sending. The name of the character set can consist of up to 40 alphanumeric characters, including appropriate punctuation. CICS does not support all the character sets named by IANA.

For conversion of the entity body to take place, the CLIENTCONV option must be specified as (or allowed to default to) CLICONVERT. Specifying NOCLICONVERT suppresses conversion of the entity body. If conversion is requested, ISO-8859-1 is used as the default if the CHARACTERSET attribute is not specified.

CHUNKING*(cvda)*

is used for controlling the message send when the message is being sent in chunks (known as chunked transfer-coding). The default when the option is not specified is that chunked transfer-coding is not in use.

The content of a chunked message can be divided into chunks in whatever way is most convenient for the application program. The body of a chunked message cannot be formed directly from CICS documents, so the DOCTOKEN option cannot be used.

Use a separate WEB SEND command with CHUNKING(CHUNKYES) for each chunk of the message. Use the FROM option to specify the chunk of data, and the FROMLENGTH option to specify the length of the chunk. Other options for the message, such as the CLOSESTATUS option, can be specified on the first WEB SEND command of the sequence (which sends the first chunk), but do not specify them on subsequent commands (which send the second and subsequent chunks).

When you have sent the last chunk of the data, specify a further WEB SEND command with CHUNKING(CHUNKEND) and no FROM or FROMLENGTH option. CICS then sends an empty chunk to the recipient to end the chunked message.

CVDA values are:

CHUNKNO

Chunked transfer-coding is not used for the message. This is the default if the CHUNKING option is not specified.

CHUNKYES

Chunked transfer-coding is in progress. The data specified by the FROM option represents a chunk of the message.

CHUNKEND

Chunked transfer-coding is complete. No data is specified for this send. CICS sends an empty chunk to the recipient to complete the chunked message.

Note:

1. The method (METHOD option) must be compatible with chunked transfer-coding.

2. When you have begun sending the parts of a chunked message, the application program cannot send any different messages or receive any items until the final empty chunk is sent and the chunked message is complete.

CLOSESTATUS*(cvda)*

specifies whether or not a Connection header with the "close" connection option (Connection: close) should be included on the message. The default is that the header is not included. The CVDA values are:

CLOSE

makes CICS write a Connection: close header for this request. The header notifies the server that the connection should be closed after the server has sent its response to the request. (For a server at HTTP/1.0 level, CICS achieves the same effect by omitting the Connection: Keep-Alive header.)

If chunked transfer-coding is in use, the CLOSESTATUS(CLOSE) option can be specified on the first chunk of the message, to inform the server that the connection should be closed after the chunked message is complete and a response has been sent.

If chunked transfer-coding is not in use, and the CLOSESTATUS(CLOSE) option is specified on a WEB SEND command, no further messages can be sent to the server until a new connection is made.

NOCLOSE

means that the Connection: close header is not used for this request. If the server is identified as HTTP/1.0, CICS sends a Connection header with the "Keep-Alive" connection option (Connection: Keep-Alive), to notify that a persistent connection is desired.

CLIENTCONV*(cvda)*

specifies whether or not CICS translates the entity body of the HTTP request before sending, from the code page used by the application, to a character set suitable for the recipient. If this option is omitted, the default is that any entity body **is** converted, unless a non-text media type is specified. CVDA values are:

CLICONVERT

CICS converts the entity body of the HTTP request from the code page used by the application, into the character set that you identify for the server. You can use the CHARACTERSET option on this command to specify the character set that is used. If conversion is requested but you do not specify a character set, the default is that CICS converts the entity body to the ISO-8859-1 character set. (The code page used by the application was identified on the WEB OPEN command for the connection.)

NOCLICONVERT

CICS does not convert the entity body of the HTTP request, and it is

sent to the server in the code page used by the application, as identified on the WEB OPEN command for the connection.

DOCTOKEN(*data-value*)

specifies the 16-byte binary token of a document to be sent as the message body. The document must be created using the CICS Document interface (EXEC CICS DOCUMENT CREATE, INSERT, and SET commands), as described in the *CICS Application Programming Guide*. The FROM option provides an alternative way to create a message body.

The body of a chunked message cannot be formed from CICS documents, so the DOCTOKEN option cannot be used for chunked transfer-coding.

CICS documents cannot be converted to the UTF-8 and UTF-16 character encodings.

FROM(*data-area*)

specifies a buffer of data which holds the message body. The message body is built by the application program. When you specify the FROM option, use the FROMLENGTH option to specify the length of the buffer of data. The DOCTOKEN option provides an alternative way to create the message body, but that option cannot be used for the body of a chunked message.

FROMLENGTH(*data-value*)

specifies the length, as a fullword binary value, of the buffer of data supplied on the FROM option (the message body). It is important to state this value correctly, because an incorrect data length can cause problems for the recipient of the message.

MEDIATYPE(*data-value*)

specifies the data content of any message body provided, for example `text/xml`. The media type is up to 56 alphanumeric characters, including
\# appropriate punctuation, but not spaces. CICS checks that the format of the media type is correct, but does not check the validity of the media type against
\# the data content. CICS uses this information to produce the Content-Type
\# header for the message.

\# For requests which require a body, you must specify the MEDIATYPE option.
\# There is no default. However, if the required Content-Type header needs to
\# contain spaces or more than 56 characters, the application can provide it using
\# the WEB WRITE HTTPHEADER command. In this case, do not specify the
\# MEDIATYPE option.

METHOD(*cvda*)

specifies the HTTP method for the request.

The GET, HEAD, POST, PUT, TRACE, OPTIONS, and DELETE methods are supported by this command. However, some HTTP servers, particularly HTTP/1.0 servers, might not implement all of these methods.

CICS bars the sending of a message body for methods where it is inappropriate, and requires it for methods where it is appropriate. Chunked transfer-coding is not relevant for methods that do not have a request body.The CVDA values are:

GET   Obtain a resource from the server. A request body is not allowed.

HEAD Obtain the HTTP headers, but not the response body, for a resource. A request body is not allowed.

POST Send data to a server. A request body is required.

PUT   Create or modify a resource on the server. A request body is required.

TRACE

Trace the route of your request to the server. A request body is not allowed.

OPTIONS

Obtain information about the server. A request body is allowed, but there is no defined purpose for the body. If you do use a request body, then you must specify a media type.

DELETE

Delete a resource on the server. A request body is not allowed.

PATH*(data-area)*

specifies the path information for the specific resource within the server that the application needs to access.

If the URIMAP option was used to specify an existing URIMAP definition on the WEB OPEN command for this connection, the path specified in that URIMAP definition is the default path for the WEB SEND command. In these circumstances, if you do not specify path information on the WEB SEND command, the path from the URIMAP definition is used. If you specify a different path from that given in the URIMAP definition, this overrides the path from the URIMAP definition.

If the URIMAP option was not used on the WEB OPEN command, there is no default path, and you must provide path information. Path information can be extracted from a known URL using the WEB PARSE URL command.

As an alternative to using the PATH option to provide the path information, you can use the URIMAP option on the WEB SEND command to specify a URIMAP definition from which the path information is taken directly.

PATHLENGTH*(data-value)*

specifies the length of the path, as a fullword binary value. If you are providing path information using the PATH option, you need to specify the PATHLENGTH option. Path length information is returned if you use the WEB PARSE URL command to parse a URL.

QUERYSTRING*(data-area)*

specifies a query string that is to be supplied to the server as part of the request. You do not need to include a question mark (?) at the beginning of the query string; if you do not include it, CICS supplies it for you automatically when constructing the request. If you include escaped characters in the query string, CICS passes them to the server in their escaped format.

QUERYSTRLEN*(data-value)*

specifies the length of the query string supplied on the QUERYSTRING option, as a fullword binary value.

SESSTOKEN*(data-value)*

specifies the session token, an 8-byte binary value that uniquely identifies a connection between CICS and a server. This value is returned by a WEB OPEN command for CICS as an HTTP client.

URIMAP*(data-value)*

specifies the name (up to eight characters, in mixed case) of a URIMAP definition that provides the path information for the specific resource within the server that the application needs to access. The URIMAP definition must be for CICS as an HTTP client (with USAGE(CLIENT) specified). Its HOST attribute must be the same as the HOST attribute of the URIMAP definition that was specified on the WEB OPEN command for this connection, or the same as the

host name specified in the HOST option on the WEB OPEN command for this connection. A URIMAP definition specified on the WEB SEND command applies only to this request.

If the URIMAP option is specified, do not specify the PATH or PATHLENGTH options.

WEB RECEIVE (Client)

Receive an HTTP response for CICS as an HTTP client.

WEB RECEIVE (CICS as an HTTP client)

```
►►──WEB──RECEIVE──SESSTOKEN(data-value)───────────────────────────────►
                                        └─MEDIATYPE(data-area)─┘

►──────────────────────────────────────────────────────────────────►
   └─STATUSCODE(data-value)─┘

►───────────────────────────────────────────────INTO(data-area)────►
   └─STATUSTEXT(data-area)──STATUSLEN(data-value)─┘  └─SET(ptr-ref)──┘

►─LENGTH(data-area)──────────────────────────────────────────────────►
                   └─MAXLENGTH(data-value)─┘  └─NOTRUNCATE─┘

►───────────────────────────────────────────────────────────────────►◄
   └─CLIENTCONV(cvda)─┘
```

Conditions: INVREQ, LENGERR, NOTOPEN, IOERR, TIMEDOUT

This command is threadsafe.
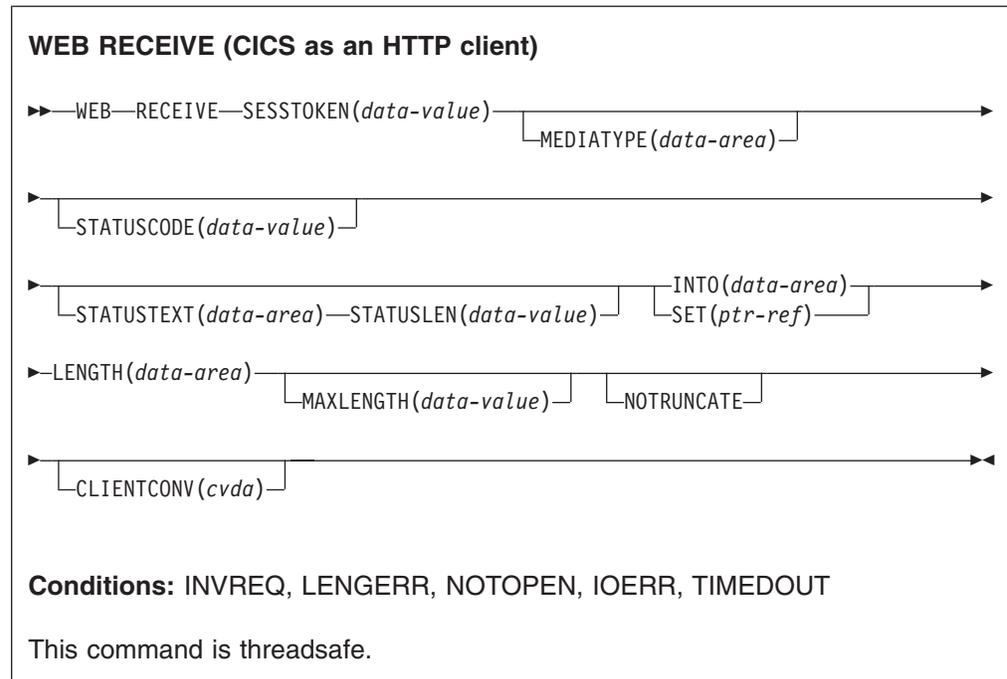
Description

WEB RECEIVE for CICS as an HTTP client receives the body of an HTTP response that a server has made. The headers for the HTTP response can be examined separately using the WEB READ HTTPHEADER command or the HTTP header browsing commands. A session token must be included on this command.

Code page conversion for the incoming message can be specified on this command.

Note: The RTIMOUT value specified for the transaction that starts the user application indicates the time that the application is prepared to wait to receive the incoming message. (RTIMOUT is specified on the transaction profile definition). When the period specified by RTIMOUT expires, CICS returns a TIMEDOUT response to the application. An RTIMOUT value of zero means that the application is prepared to wait indefinitely. The default setting for RTIMOUT on transaction profile definitions is zero, so it is important to check and change that setting for applications that are making HTTP client requests.

Tip: For CICS as an HTTP client, the CONVERSE command can be used as an alternative to issuing a WEB SEND command followed by a WEB RECEIVE command.

Options

CLIENTCONV*(cvda)*

    specifies whether or not CICS translates the entity body of the response from the character set used by the server, to a code page suitable for the application. The default is that the entity body is converted.

    CLICONVERT

        CICS converts the entity body of the response from the character set used by the server, into the code page that you identify for the application.

    NOCLICONVERT

        CICS does not convert the entity body of the response, and it is passed to the application in the character set used by the server.

    You do not need to specify a character set or application code page on the WEB RECEIVE command for CICS as an HTTP client. If code page conversion is required, CICS identifies the character set used by the server by examining the Content-Type header of the message. If the header does not provide this information, or if the named character set is not supported by CICS for code page conversion, the ISO-8859-1 character set is used. For the application's code page, the default code page for the local CICS region (as specified in the LOCALCCSID system initialization parameter) is used, or an alternative EBCDIC code page that you specified on the WEB OPEN COMMAND.

INTO*(data-area)*

    specifies the buffer that is to contain the data being received.

LENGTH*(data-area)*

    specifies a fullword binary variable which is set to the amount of data that CICS has returned to the application.

- If the NOTRUNCATE option **is not** specified, any further data present in the message has now been discarded. A LENGERR response with a RESP2 value of 57 is returned if further data was present.
- If the NOTRUNCATE option **is** specified, any additional data is retained. A LENGERR response with a RESP2 value of 36 is returned if additional data is available. The description for the NOTRUNCATE option tells you what to do in this case.

MAXLENGTH*(data-value)*

    specifies the maximum amount, as a fullword binary value, of data that CICS is to pass to the application. The MAXLENGTH option applies whether the INTO or the SET option is specified for receiving data. If the data has been sent using chunked transfer-coding, CICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the chunked message, rather than to each individual chunk. The data is measured after any code page conversion has taken place.

MEDIATYPE*(data-area)*

    specifies a 56-character data-area to receive the media type (that is, the type of data content) for the body, for example `text/xml`.

NOTRUNCATE

    specifies that when the data available exceeds the length requested on the MAXLENGTH option, the remaining data is not to be discarded immediately but is to be retained for retrieval by subsequent RECEIVE commands. (If no further RECEIVE commands are issued, the data is discarded during transaction termination.)

A single RECEIVE command using the SET option and without the
MAXLENGTH option receives all the remaining data, whatever its length.
Alternatively, you can use a series of RECEIVE commands with the
NOTRUNCATE option to receive the remaining data in appropriate chunks.
Keep issuing the RECEIVE command until you are no longer getting a
LENGERR response.

SET*(ptr-ref)*
specifies a pointer reference that is to be set to the address of data received.
The pointer reference is valid until the next receive command or the end of
task.

SESSTOKEN*(data-value)*
specifies the session token, an 8-byte binary value that uniquely identifies a
connection between CICS and a server. This value is returned by a WEB OPEN
command for CICS as an HTTP client.

STATUSCODE*(data-value)*
specifies a data-area to receive the HTTP status code sent by the server. The
code is a binary halfword value. Examples are 200 (normal) or 404 (not found).
Receiving the status code is optional, but you should always receive and check
the status code in the following circumstances:

- If you intend to make an identical request to the server, now or during a
  future connection.
- If you intend to make further requests to the server using this connection.
- If your application is carrying out any further processing that depends on the
  information you receive in the response.

STATUSTEXT*(data-area)*
specifies a data-area to receive any text returned by the server to describe the
status code. The text is known as a reason phrase. Examples are "OK"
(accompanying a 200 status code), or "Bad Request" (accompanying a 400
status code). The STATUSLEN option gives the length allowed for the text.

STATUSLEN*(data-value)*
specifies, as a fullword binary value, the length of the data-area to receive any
text returned by the server to describe the status code (the STATUSTEXT
option). The text is known as a reason phrase. Most reason phrases
recommended for HTTP are short, but a data-area length of 256 characters is
suggested here, in case the server replaces the recommended reason phrase
with more detailed information.

WEB CONVERSE
Send an HTTP request by CICS as an HTTP client, and receive a response from
the server, using a single command. An alternative to the WEB SEND and WEB
RECEIVE commands for CICS as an HTTP client.

WEB CONVERSE

```
►►──WEB──CONVERSE──SESSTOKEN(data-value)─────────────────────────────────►

   ┌─────────────────────────────────────────────────┐  ┌──────────────┐
►──┤                                                 ├──┤METHOD(cvda)  ├──►
   ├─PATH(data-area)──PATHLENGTH(data-value)─┤
   └─URIMAP(data-value)──────────────────────┘

►─────┬─────────────────────────┬──────────────────────────────────────►
      └─MEDIATYPE(data-area)─────┘

►─────┬──────────────────────────────────────────────┬─────────────────►
      └─QUERYSTRING(data-area)──QUERYSTRLEN(data-value)┘

►─────┬─────────────────────────────────────────┬──┬────────────────┬──►
      ├─DOCTOKEN(data-value)────────────────────┤  └─ACTION(cvda)────┘
      └─FROM(data-area)──FROMLENGTH(data-value)─┘

                                ┌─INTO(data-area)─┐  ┌─TOLENGTH(data-area)─┐
►─────┬─────────────────────┬───┤                 ├──┤                     ├──►
      └─CLOSESTATUS(cvda)───┘   └─SET(ptr-ref)────┘

►─────┬──────────────────────┬──┬───────────┬──┬─────────────────────┬──►
      └─MAXLENGTH(data-value)┘  └─NOTRUNCATE─┘  └─STATUSCODE(data-area)┘

►─────┬────────────────────────────────────────────┬───────────────────►
      └─STATUSTEXT(data-area)──STATUSLEN(data-value)┘

►─────┬──────────────────────────┬──┬──────────────────┬──────────────►◄
      └─CHARACTERSET(data-value)─┘  └─CLIENTCONV(cvda)──┘
```

Conditions: IOERR, INVREQ, LENGERR, NOTAUTH, NOTFND, NOTOPEN, TIMEDOUT, TOKENERR

This command is threadsafe.

Description

WEB CONVERSE enables an application program to compose and send an HTTP client request, and receive a response from the server. A session token must be included on this command.

- **The HTTP client request** is made using a connection that has been opened using the WEB OPEN command. The WEB CONVERSE command can be used in place of the WEB SEND command to compose and send the request.
- **The response from the server** is received by CICS Web support and passed to the application. The WEB CONVERSE command can be used in place of the WEB RECEIVE command to make the application program wait for and receive the HTTP response. The headers for the HTTP response can be examined separately using the WEB READ HTTPHEADER command or the HTTP header browsing commands.

> **Note:** The RTIMOUT value specified for the transaction that starts the user application indicates the time that the application is prepared to wait to receive the incoming message. (RTIMOUT is specified on the transaction profile definition). When the period specified by RTIMOUT expires, CICS returns a TIMEDOUT response to the application. An RTIMOUT value of

zero means that the application is prepared to wait indefinitely. The default setting for RTIMOUT on transaction profile definitions is zero, so it is important to check and change that setting for applications that are making HTTP client requests.

The WEB CONVERSE command does not support chunked transfer-coding for the request, because this requires a sequence of send actions, and the WEB CONVERSE command provides a single send action. If you want to send a chunked message, use the WEB SEND command to send it, and the WEB RECEIVE command to receive it. If the server sends a chunked response, this can be received using the WEB CONVERSE command.

The WEB CONVERSE command cannot be used after the connection to the server has been closed. If you need to test whether the server has requested termination of the connection, use the WEB READ HTTPHEADER command to look for the Connection: close header in the last message from the server.

The WEB CONVERSE command performs a single send action and a single receive action, and it is designed to be used in place of a WEB SEND command and a WEB RECEIVE command. You may use WEB SEND and WEB RECEIVE commands, and WEB CONVERSE commands, in relation to the same connection (that is, with the same SESSTOKEN). However, if you are pipelining requests (that is, sending a sequence of requests without waiting for a response), you must not follow a WEB SEND command with a WEB CONVERSE command. CICS checks at program run time that each WEB SEND command has a subsequent WEB RECEIVE command before any WEB CONVERSE command is issued. For example, if you use the WEB SEND command three times to issue a pipelined sequence of requests, you must use the WEB RECEIVE command three times to receive the responses for those requests, before you may use the WEB CONVERSE command.

Options for sending the HTTP client request

ACTION*(cvda)*
> This option is used to specify how the message should be sent out. The CVDA value that applies for CICS as an HTTP client is:

> **EXPECT**
>> makes CICS send an Expect header along with the request line and headers for the request, and await a 100-Continue response before sending the message body to the server. If a response other than 100-Continue is received, CICS informs the application program and cancels the send. If no response is received after a period of waiting, CICS sends the message body anyway.

>> This option must only be used if your request has a message body.

CLOSESTATUS*(cvda)*
> specifies whether or not a Connection header with the "close" connection option (Connection: close) should be included on the request. The default is that the header is not included. The CVDA values are:

> **CLOSE**
>> makes CICS write a Connection: close header for this request. The header notifies the server that the connection should be closed after the server has sent its response to the request. (For a server at HTTP/1.0 level, CICS achieves the same effect by omitting the Connection: Keep-Alive header.)

NOCLOSE

>> means that the Connection: close header is not used for this request. If the server is identified as HTTP/1.0, CICS sends a Connection header with the "Keep-Alive" connection option (Connection: Keep-Alive), to notify that a persistent connection is desired.

DOCTOKEN*(data-value)*

specifies the 16-byte binary token of a document to be sent as the message body. The document must be created using the CICS Document interface (EXEC CICS DOCUMENT CREATE, INSERT, and SET commands), as described in the *CICS Application Programming Guide*. The FROM option provides an alternative way to create a message body.

CICS documents cannot be converted to the UTF-8 and UTF-16 character encodings.

FROM*(data-area)*

specifies a buffer of data which holds the message body. The message body is built by the application program. When you specify the FROM option, use the FROMLENGTH option to specify the length of the buffer of data. The DOCTOKEN option provides an alternative way to create the message body.

FROMLENGTH*(data-area)*

specifies the length, as a fullword binary value, of the buffer of data supplied on the FROM option (the message body). It is important to state this value correctly, because an incorrect data length can cause problems for the recipient of the message.

MEDIATYPE*(data-area)*

specifies the data content of any request body, for example `text/xml`. The media type is up to 56 alphanumeric characters, including appropriate punctuation, but not spaces. CICS checks that the format of the media type is correct, but does not check the validity of the media type against the data content. CICS uses this information to produce the Content-Type header for the message.

For requests which require a body, you must specify the MEDIATYPE option. There is no default. However, if the required Content-Type header needs to contain spaces or more than 56 characters, the application can provide it using the WEB WRITE HTTPHEADER command. In this case, do not specify the MEDIATYPE option.

The MEDIATYPE option is used for both the sending and receiving functions of the WEB CONVERSE command. If it is specified with a value, the value is used to construct the Content-Type header in the request, and the same field is used to receive the media type of the response that is returned by the server. If it is specified without a value, it is used only to receive the media type of the response.

METHOD*(cvda)*

specifies the HTTP method for the request.

The GET, HEAD, POST, PUT, TRACE, OPTIONS, and DELETE methods are supported by this command. However, some HTTP servers, particularly HTTP/1.0 servers, might not implement all of these methods.

CICS bars the sending of a message body for methods where it is inappropriate, and requires it for methods where it is appropriate.The CVDA values are:

GET Obtain a resource from the server. A request body is not allowed.

HEAD Obtain the HTTP headers, but not the response body, for a resource. A request body is not allowed.

POST Send data to a server. A request body is required.

PUT Create or modify a resource on the server. A request body is required.

TRACE
Trace the route of your request to the server. A request body is not allowed.

OPTIONS
Obtain information about the server. A request body is allowed, but there is no defined purpose for the body. If you do use a request body, then you must specify a media type.

DELETE
Delete a resource on the server. A request body is not allowed.

PATH*(data-area)*
specifies the path information for the specific resource within the server that the application needs to access.

If the URIMAP option was used to specify an existing URIMAP definition on the WEB OPEN command for this connection, the path specified in that URIMAP definition is the default path for the WEB SEND command. In these circumstances, if you do not specify path information on the WEB SEND command, the path from the URIMAP definition is used. If you specify a different path from that given in the URIMAP definition, this overrides the path from the URIMAP definition.

If the URIMAP option was not used on the WEB OPEN command, there is no default path, and you must provide path information. Path information can be extracted from a known URL using the WEB PARSE URL command.

As an alternative to using the PATH option to provide the path information, you can use the URIMAP option on the WEB CONVERSE command to specify a URIMAP definition from which the path information is taken directly.

PATHLENGTH*(data-value)*
specifies the length of the path, as a fullword binary value. If you are providing path information using the PATH option, you need to specify the PATHLENGTH option. Path length information is returned if you use the WEB PARSE URL command to parse a URL.

QUERYSTRING*(data-area)*
specifies a query string that is to be supplied to the server as part of the request. You do not need to include a question mark (?) at the beginning of the query string; if you do not include it, CICS supplies it for you automatically when constructing the request. If you include escaped characters in the query string, CICS passes them to the server in their escaped format.

QUERYSTRLEN*(data-value)*
specifies the length of the query string supplied on the QUERYSTRING option, as a fullword binary value.

SESSTOKEN*(data-value)*
specifies the session token, an 8-byte binary value that uniquely identifies this connection between CICS and a server. This value is returned by a WEB OPEN command for CICS as an HTTP client. "Session tokens" in the *CICS Internet Guide* explains the use of the session token.

URIMAP(*data-value*)
specifies the name (up to 8 characters, in mixed case) of a URIMAP definition that provides the path information for the specific resource within the server that the application needs to access. The URIMAP definition must be for CICS as an HTTP client (with USAGE(CLIENT) specified). Its HOST attribute must be the same as the HOST attribute of the URIMAP definition that was specified on the WEB OPEN command for this connection, or the same as the host name specified in the HOST option on the WEB OPEN command for this connection. A URIMAP definition specified on the WEB CONVERSE command applies only to this request.

If the URIMAP option is specified, do not specify the PATH or PATHLENGTH options.

Options for receiving the server's response

INTO(*data-area*)
specifies the buffer that is to contain the data being received.

MAXLENGTH(*data-value*)
specifies the maximum amount, as a fullword binary value, of data that CICS is to pass to the application. The MAXLENGTH option applies even when the INTO option is specified for receiving data. If the data has been sent using chunked transfer-coding, CICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the chunked message, rather than to each individual chunk. The data is measured after any code page conversion has taken place.

MEDIATYPE(*data-area*)
specifies a 56-character data-area to receive the media type (that is, the type of data content) for the body, for example `text/xml`.

The MEDIATYPE option is used for both the sending and receiving functions of the WEB CONVERSE command. If it is specified with a value, the value is used to construct the Content-Type header in the request, and the same field is used to receive the media type of the response that is returned by the server. If it is specified without a value, it is used only to receive the media type of the response.

NOTRUNCATE
specifies that when the data available exceeds the length requested on the MAXLENGTH option, the remaining data is not to be discarded immediately but is to be retained for retrieval by subsequent RECEIVE commands. (If no further RECEIVE commands are issued, the data is discarded during transaction termination.)

A single RECEIVE command using the SET option and without the MAXLENGTH option receives all the remaining data, whatever its length. Alternatively, you can use a series of RECEIVE commands with the NOTRUNCATE option to receive the remaining data in appropriate chunks. Keep issuing the RECEIVE command until you are no longer getting a LENGERR response.

SET(*ptr-ref*)
specifies a pointer reference that is to be set to the address of data received. The pointer reference is valid until the next receive command or the end of task.

STATUSCODE(*data-area*)
specifies a data-area to receive the HTTP status code sent by the server. The

code is a binary halfword value. Examples are 200 (normal) or 404 (not found). Receiving the status code is optional, but you should always receive and check the status code in the following circumstances:

- If you intend to make an identical request to the server, now or during a future connection.
- If you intend to make further requests to the server using this connection.
- If your application is carrying out any further processing that depends on the information you receive in the response.

STATUSTEXT*(data-area)*
specifies a data-area to receive any text returned by the server to describe the status code. The text is known as a reason phrase. Examples are "OK" (accompanying a 200 status code), or "Bad Request" (accompanying a 400 status code). The STATUSLEN option gives the length allowed for the text.

STATUSLEN*(data-value)*
specifies, as a fullword binary value, the length of the data-area to receive any text returned by the server to describe the status code (the STATUSTEXT option). The text is known as a reason phrase. Most reason phrases recommended for HTTP are short, but a data-area length of 256 characters is suggested here, in case the server replaces the recommended reason phrase with more detailed information.

TOLENGTH*(data-area)*
specifies a fullword binary variable which is set to the amount of data that CICS has returned to the application.

- If the NOTRUNCATE option **is not** specified, any further data present in the message has now been discarded. A LENGERR response with a RESP2 value of 57 is returned if further data was present.
- If the NOTRUNCATE option **is** specified, any additional data is retained. A LENGERR response with a RESP2 value of 36 is returned if additional data is available. The description for the NOTRUNCATE option tells you what to do in this case.

This option is the equivalent of the LENGTH option on the WEB RECEIVE command.

Options for converting items sent and received

CHARACTERSET*(data-value)*
specifies the character set into which CICS translates the entity body of the HTTP request before sending. The name of the character set can consist of up to 40 alphanumeric characters, including appropriate punctuation. CICS does not support all the character sets named by IANA.

For conversion of the request body to take place, the CLIENTCONV option must be allowed to default to CLICONVERT, or specified as NOINCONVERT. Specifying NOCLICONVERT or NOOUTCONVERT suppresses conversion of the request body. If conversion is requested, ISO-8859-1 is used as the default if the CHARACTERSET attribute is not specified.

CLIENTCONV*(cvda)*
specifies whether or not CICS translates the entity body of the HTTP request before sending, and translates the entity body of the server's response. The default is that the entity body **is** converted both when the request is sent out, and when the response is received (CLICONVERT).

- For the request body, you can use the CHARACTERSET option on this command to specify a character set that is suitable for the server. If

conversion is requested (or happens as the default) but you do not specify a character set, the default is that CICS converts the entity body to the ISO-8859-1 character set.

- For the response body, you do not need to specify the character set used by the server. CICS identifies this by examining the Content-Type header of the message. If the header does not provide this information, or if the named character set is not supported by CICS for code page conversion, the ISO-8859-1 character set is used.

- For the application's code page, the default code page for the local CICS region (as specified in the LOCALCCSID system initialization parameter) is used, or an alternative EBCDIC code page that you specified on the WEB OPEN COMMAND.

CVDA values are:

CLICONVERT
> CICS converts the entity body of the request into the character set that you identify for the server, and converts the entity body of the response into a code page suitable for the application.

NOINCONVERT
> CICS converts the entity body of the request into the character set that you identify for the server. However, CICS does **not** convert the entity body of the response, and it is passed to the application in the character set used by the server.

NOOUTCONVERT
> CICS does **not** convert the entity body of the request, and it is sent to the server in the code page used by the application. However, CICS does convert the entity body of the response into a code page suitable for the application.

NOCLICONVERT
> CICS does not convert the entity body of the request, and it is sent to the server in the code page used by the application. CICS does not convert the entity body of the response, and it is passed to the application in the character set used by the server.

Changes to options on `EXEC CICS WEB` commands

The SESSTOKEN option is added to the `EXEC CICS WEB` commands for HTTP headers, so that the commands can be used for CICS as an HTTP client.

The SESSTOKEN option is added to the following commands:
- WEB WRITE HTTPHEADER
- WEB READ HTTPHEADER
- WEB STARTBROWSE HTTPHEADER
- WEB READNEXT HTTPHEADER
- WEB ENDBROWSE HTTPHEADER

`SESSTOKEN`*(data-value)*
> For CICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between CICS and a server. This value is returned by a WEB OPEN command for CICS as an HTTP client."Session tokens" on page 85 explains the use of the session token.

Changes to the JCICS API

Note: To get this function, apply the PTF for APAR PK04303.

The JCICS application programming interface is extended to support the client functions of the following CICS commands:

    WEB OPEN
    WEB CLOSE
    WEB SEND
    WEB RECEIVE
    WEB CONVERSE
    WEB READ HTTPHEADER
    WEB WRITE HTTPHEADER
    WEB STARTBROWSE HTTPHEADER
    WEB READNEXT HTTPHEADER
    WEB ENDBROWSE HTTPHEADER

Changes to global user exits

There are two new global user exits for CICS as an HTTP client: XWBOPEN in the WEB OPEN command, and XWBSNDO in the WEB SEND command. (Note that XWBSNDO only applies when the WEB SEND command is used for CICS as an HTTP client, and not for CICS as an HTTP server.)

HTTP client open exit XWBOPEN

XWBOPEN enables systems administrators to specify proxy servers that should be used for HTTP requests by CICS as an HTTP client, and to apply a security policy to the host name specified for those requests. XWBOPEN is called during processing of an **EXEC CICS** WEB OPEN command, which is used by an application program to open a connection with a server.

The **EXEC CICS** WEB OPEN command instructs the CICS Web domain to open a connection with a server. XWBOPEN is called before the connection is opened. The host name for the connection (for example, `www.example.com`), which is specified by the HOST option on the **EXEC CICS** WEB OPEN command, is passed as the UEPHOST parameter to the user exit program for checking. At this point, the user exit program can be used for two purposes:

- **To determine whether the HTTP request needs to use a proxy server, and return the name of any proxy server that is required.** If a proxy server is needed, return code UERCPROX is used, and the name of the proxy server is returned to the CICS Web domain (in the buffer identified by UEPPROXY) and used to make the connection to the server. If no proxy server is needed, return code UERCNORM is used.

- **To apply a security policy to the host name.** Return code UERCBARR indicates that access to the host is not permitted. If access to the host is not permitted, an INVREQ response is returned to the WEB OPEN command, and the application programmer should abandon the attempt to open that connection. If you want to apply a security policy for individual resources, as well as (or instead of) for the host, the XWBSNDO user exit on the **EXEC CICS** WEB SEND and **EXEC CICS** WEB CONVERSE commands can be used to apply a security policy to the path component of the URL.

The XWBOPEN user exit does not support the use of `EXEC CICS` commands.

The sample programs DFH$WBPI and DFH$WBEX, and the associated copybook DFH$WBGA, show you how to set up proxy server information or a security policy in a global work area. For example, if all the requests from your CICS system should use a single proxy server, you can specify the proxy server name as an initialization parameter. If you use a number of proxy servers or want to apply a security policy to different host names, you could load or build a table that matches host names to appropriate proxy servers or marks them as barred, which could then be used as a look-up table during processing of the `EXEC CICS` WEB OPEN command. The sample programs can be run during program list table post initialization (PLTPI) processing, or at any point before you expect the `EXEC CICS` WEB OPEN command to be used.

Exit XWBOPEN

When invoked

During processing of an `EXEC CICS` WEB OPEN command.

Exit-specific parameters

UEPHOST (Input supplied by CICS)

The address of a field containing the host name specified in the HOST option of the WEB OPEN command.

Note: The host name is converted into lower case when it is saved in this field. Your user exit program should take this into account when matching the host name.

UEPHOSTL (Input supplied by CICS)

The address of a field containing the halfword length of the host name.

UEPPROXY (Output supplied by user exit)

The address of a field containing the address that points to the proxy server name. On input to the user exit program, the parameter is set to the address of a field containing the address of a 2046-byte area. You can place the proxy server name in this area, and leave the address in UEPPROXY unchanged. Alternatively, you can place the proxy server name in your own area, and replace the address in UEPPROXY with the address of a field containing the address of your own area.

UEPPROXYL (Output supplied by user exit)

The address of a field containing the halfword length of the proxy server name.

Return codes

UERCNORM

A proxy server is not needed for this HTTP request, and the host name is not barred.

UERCPROX

A proxy server is needed for this HTTP request. UEPPROXY has been set to the name of the required proxy server, and UEPPROXYL has been set to the length of the proxy server name.

UERCBARR

The host name of the server is barred.

UERCERR

An error occurred in exit processing.

XPI calls

All XPI calls can be used.

API and SPI commands

No `EXEC CICS` commands can be used.

HTTP client send exit XWBSNDO

XWBSNDO enables systems administrators to specify a security policy for HTTP requests by CICS as an HTTP client. XWBSNDO is called during processing of an `EXEC CICS` WEB SEND or `EXEC CICS` WEB CONVERSE command. The host name and path information are passed to the exit, and a security policy can be applied to either or both of these components.

The XWBOPEN exit on the WEB OPEN command can be used to bar access to a whole host, and the XWBSNDO exit can be used to do the same or to bar access to specific paths within a host. If you want to bar access to a whole host, doing this with the XWBOPEN exit saves time, because the application program is not able to open the connection and so does not waste time creating the request that should be sent. The host name is provided to the XWBSNDO exit with the primary intention of allowing you to differentiate between identical paths used by different hosts.

If chunked transfer-coding is being used for the HTTP request, XWBSNDO is only called on the first WEB SEND command for the chunked message.

The XWBSNDO user exit does not support the use of `EXEC CICS` commands.

The host is passed to the user exit program as the UEPHOST parameter, and the path is passed as the UEPPATH parameter. Return code UERCNORM indicates that the path is permitted, and return code UERCBARR indicates that the path is not permitted. If the path is not permitted, an INVREQ response is returned to the WEB SEND or WEB CONVERSE command, and the application programmer should handle this by closing the connection with a WEB CLOSE command.

Exit XWBSNDO

When invoked

During processing of an `EXEC CICS` WEB SEND or `EXEC CICS` WEB CONVERSE command for an HTTP request by CICS as an HTTP client. A client request is indicated by the use of the SESSTOKEN parameter on the WEB SEND command.

Exit-specific parameters

UEPHOST

The address of a field containing the host name specified in the HOST option of the WEB OPEN command for the connection.

Note: The host name is converted into lower case when it is saved in this field. Your user exit program should take this into account when matching the host name.

UEPHOSTL

The address of a field containing the halfword length of the host name.

UEPPATH

The address of a field containing the path specified in the PATH option of the WEB SEND command. The path is in mixed case, as it was specified.

UEPPATHL
The address of a field containing the halfword length of the path.

Return codes

UERCNORM
The path is permitted.

UERCBARR
The path is not permitted.

XPI calls
All XPI calls can be used.

API and SPI commands
No **EXEC CICS** commands can be used.

Changes to monitoring

The following fields are added to the DFHWEBB group of performance-class monitoring records:

331 (TYPE-A, 'WBREDOCT', 4 BYTES)
The number of CICS Web support READ HTTPHEADER requests issued by the user task when CICS is an HTTP client.

332 (TYPE-A, 'WBWRTOCT', 4 BYTES)
The number of CICS Web support WRITE HTTPHEADER requests issued by the user task when CICS is an HTTP client.

333 (TYPE-A, 'WBRCVIN1', 4 BYTES)
The number of CICS Web support RECEIVE and CONVERSE requests issued by the user task when CICS is an HTTP client.

334 (TYPE-A, 'WBCHRIN1', 4 BYTES)
The number of bytes received by the CICS Web support RECEIVE and CONVERSE requests issued by the user task when CICS is an HTTP client. This includes the HTTP headers for the response.

335 (TYPE-A, 'WBSNDOU1, 4 BYTES)
The number of CICS Web support SEND and CONVERSE requests issued by the user task when CICS is an HTTP client.

336 (TYPE-A, 'WBCHROU1', 4 BYTES)
The number of bytes sent by the CICS Web support SEND and CONVERSE requests issued by the user task when CICS is an HTTP client. This includes the HTTP headers for the request.

337 (TYPE-A, 'WBPARSCT', 4 BYTES)
The number of CICS Web support PARSE URL requests issued by the user task.

338 (TYPE-A, 'WBBRWOCT', 4 BYTES)
The number of CICS Web support BROWSE HTTPHEADER requests (STARTBROWSE, READNEXT, and ENDBROWSE) issued by the user task when CICS is an HTTP client.

Note: When requests are made using the WEB CONVERSE command, this increments both the Send and Receive request counts (WBSNDOU1 and WBRCVIN1) and the counts of characters sent and received (WBCHRIN1 and WBCHROU1).

Chapter 4. CICS Web support upgrade to HTTP/1.1

CICS Web support now supports HTTP/1.1.

Releases of CICS before CICS Transaction Server for z/OS, Version 3 Release 1 supported HTTP/1.0. CICS Web support is now enhanced to handle and provide features of the HTTP/1.1 specification, including chunked transfer-coding, pipelining, and persistent connections.

CICS Web support is conditionally compliant with the HTTP/1.1 specification, as described in the Internet Society and IETF (Internet Engineering Task Force) Request for Comments document RFC 2616, *Hypertext Transfer Protocol - HTTP/1.1*. (RFC 2616 is available to download from http://www.ietf.org/rfc/rfc2616.txt.) Conditional compliance with the HTTP/1.1 specification means that CICS satisfies all the "MUST" level requirements, but not all the "SHOULD" level requirements, that are detailed in the HTTP/1.1 specification, where these requirements are relevant to the functions actually provided by CICS itself. Your user application programs share the responsibility for compliance in the actions that they perform, and guidance is provided to help you make your application programs compliant, when you are ready to do that.

New behavior for CICS TS Version 3

- CICS checks inbound messages for compliance with HTTP/1.1, and handles or rejects non-compliant messages.
- CICS follows the HTTP/1.1 rules for comparison of URLs.
- CICS provides a suitable HTTP version number in the start line of outbound messages.
- On outbound HTTP/1.1 messages, CICS supplies the HTTP headers that should normally be present for the message to be compliant with HTTP/1.1:
  - Content-Length (for client request or server response).
  - Content-Type (for client request or server response).
  - Date (for client request or server response).
  - Host (for client request).
  - Last-Modified (for static server response with HFS file only).
  - Server (for server response).
  - TE (for client request).
  - Transfer-Encoding (for client request or server response).
  - User-Agent (for client request).
- CICS takes action on the Expect header for both inbound and outbound requests.
- CICS handles OPTIONS requests from Web clients and makes an appropriate response.
- CICS handles TRACE requests from Web clients and makes an appropriate response.
- CICS accepts inbound messages with chunked transfer-coding and assembles them for you, and supports your use of chunked transfer-coding to send outbound messages.
- CICS supports pipelining for both inbound and outbound messages.
- CICS supports virtual hosting (multiple host names at the same IP address), based on your URIMAP definitions.

Changed behavior, compared to CICS TS Version 2

- Connections are now persistent by default.
- CICS handles a wider range of error situations and unsupported messages.

HTTP functions not supported by CICS Web support

The HTTP/1.1 specification (RFC 2616) defines various roles for the parties that make use of the HTTP protocol. CICS Web support provides HTTP services that are appropriate for an origin server, for a client, and for a user agent (although a human user might not be involved for every HTTP client request).

The HTTP/1.1 specification also includes requirements that relate to roles which are not relevant to CICS Web support, and these can be ignored:

- CICS does not act as a proxy.
- CICS does not act as a gateway (an intermediary for another server) or a tunnel (a relay between HTTP connections).
- CICS does not provide caching facilities, or provide support for user-written caching facilities.
- CICS is not designed for use as a Web browser.

Benefits of CICS Web support upgrade to HTTP/1.1

The CICS Web support upgrade to HTTP/1.1 provides the following benefits:

- Conditional compliance with the HTTP/1.1 specification (RFC 2616) means that CICS Transaction Server for z/OS, Version 3 Release 1 can interact easily and correctly with a wider range of services and clients on the Internet.
- Persistent connections are now the default for connections between CICS and a server or Web client. Persistent connections improve network performance because a new connection does not have to be established for each request.
- Support for pipelining means that CICS applications can send a sequence of multiple requests to a server, without waiting for an acknowledgement after each item. CICS Web support also ensures correct handling for pipelined requests from a Web client to CICS.
- Chunked transfer-coding (known as chunking for short) allows dynamically produced content, or a large amount of content, to be transferred in convenient segments, while still enabling the recipient to verify that it has received the complete message. CICS can receive items sent in this way, and a user-written application program can send out items in this way.
- CICS automatically creates virtual hosts using your URIMAP definitions, enabling you to provide multiple host names at the same IP address.
- CICS supports requests by Web clients and by CICS applications with all the HTTP methods defined by the HTTP/1.1 specification, and also provides automatic handling and responses for requests from Web clients with OPTIONS and TRACE methods.

Requirements

There are no special hardware or software requirements to support this function.

New HTTP functionality

CICS Web support includes new areas of function.

- Chunked transfer-coding, or chunking
- Pipelining
- Persistent connections
- Virtual hosting

Chunked transfer-coding

Chunked transfer-coding, also known as chunking, involves transferring the body of a message as a series of chunks, each with its own chunk size header. The end of the message is indicated by a chunk with zero length and an empty line.

This defined process means that an application-generated entity body, or a large entity body, can be sent in convenient segments. The client or server knows the chunked message is complete when the zero length chunk is received.

The body of a chunked message can be followed by an optional trailer that contains supplementary HTTP headers, known as trailing headers. Clients and servers are not required to accept trailers, so the supplementary HTTP headers should only provide non-essential information, unless a server knows that a client accepts trailers.

To use chunked transfer-coding, both the client and server must be using HTTP/1.1. A chunked message cannot be sent to an HTTP/1.0 client. The requirements that apply to chunked transfer-coding and the use of trailing headers are defined in the HTTP/1.1 specification (RFC 2616).

How CICS Web support handles chunked transfer-coding

Messages using chunked transfer-coding can be sent and received by CICS.

CICS as an HTTP server can receive a chunked message as a request, or send one as a response. CICS as an HTTP client can send a chunked message as a request, or receive one as a response. CICS Web support handles these different cases as follows:

- When CICS as an HTTP server receives a chunked message as an HTTP request, CICS Web support recognizes the chunked encoding. It waits until all the chunks are received (indicated by the receipt of a chunk with zero length), and assembles the chunks to form a complete message. The assembled message body can be received by a user application program using the WEB RECEIVE command.
    - You can limit the total amount of data that CICS accepts for a single chunked message, using the MAXDATALEN option on the TCPIPSERVICE resource definition that relates to the port on which the request arrives.
    - When CICS is an HTTP server, the timeout value for receiving a chunked message is set by the SOCKETCLOSE attribute of the TCPIPSERVICE definition.
    - Trailing headers from the chunked message can be read using the HTTP header commands. The Trailer header identifies the names of the headers that were present as trailing headers. If you are using an analyzer program in

the processing path for the request, note that trailing headers are not passed to the analyzer program along with the main request headers.

- When CICS as an HTTP client receives a chunked message as a response to an application program's request, the chunks are also assembled before being passed to the application program as an entity body, and any trailing headers can be read using the HTTP header commands. You can specify how long the application will wait to receive the response, using the RTIMOUT attribute of the transaction profile definition for the transaction ID that relates to the application program.

- When CICS sends a chunked message, either as an HTTP server (response) or as an HTTP client (request), the application program can specify chunked transfer-coding by using the CHUNKING(CHUNKYES) option on the WEB SEND command for each chunk of the message. The message can be divided up in whatever way is most convenient for the application program. CICS sends each chunk of the message, adding appropriate HTTP headers to indicate to the recipient that chunked transfer-coding is being used. The application program issues WEB SEND with CHUNKING(CHUNKEND), to indicate the end of the message. CICS then sends an empty chunk (containing a blank line) to end the chunked message, along with any trailing headers that are wanted.

Pipelining

Pipelining involves a client sending multiple HTTP requests to a server without waiting for a response. Responses must then be returned from the server in the same sequence that the requests were received.

It is the requester's responsibility to ensure that the requests are idempotent. Idempotency means that the same result is always obtained when all, or part, of the series of requests is repeated. This ensures that if there is an error in connecting with the server, the client may retry the series of requests, even though it does not know if the server has implemented all, some, or none of the requests.

The HTTP/1.1 specification (RFC 2616) defines the rules about idempotency for HTTP requests.

If you plan on pipelining requests, check that the request sequence could be terminated at any point, and re-started from the beginning, without causing a logical error. If this is not the case, make the requests individually and await confirmation after each request.

How CICS Web support handles pipelining

A pipelined request sequence can be sent and received by CICS. CICS as an HTTP server can receive a pipelined request sequence from a Web client, and CICS as an HTTP client can send a pipelined request sequence to a server.

CICS Web support handles pipelined request sequences, and the responses to them, as follows:

- When CICS as an HTTP server receives a pipelined sequence of HTTP requests, the requests are processed serially. Each transaction handles a single request and provides a response. The remaining requests in the pipelined message sequence are held by CICS until the response to the previous request is sent, and then a new transaction is initiated to process each further request.

- When CICS as an HTTP client sends a pipelined request sequence, pipelining is enabled automatically. Each HTTP request is sent immediately, so the application

program can send multiple HTTP requests before it receives any response. When the last message in the pipelined sequence has been sent, the application can begin to receive the responses.

- When CICS as an HTTP client receives HTTP responses to a pipelined request sequence, the responses are returned to the application program in the order that CICS receives them from the server.

For CICS as an HTTP client, it is the application program's responsibility to ensure that any pipelined sequence of requests is idempotent.

Persistent connections

Persistent connections are connections between a Web client and a server that can be reused for more than one exchange of a request and a response.

In HTTP/1.0, the default action for the server was to close the connection when it had received a request from the Web client and sent a response. If the Web client wanted the server to keep the connection open, it had to send a Connection: Keep-Alive header on the request.

For HTTP/1.1, persistent connections are the default. When a connection is made between a Web client and a server, the server should keep the connection open by default. The connection should only be closed if the Web client requests closure by sending a Connection: close header, or if the server's timeout setting is reached, or if the server encounters an error.

Persistent connections improve network performance because a new connection does not have to be established for each request. Establishing a new connection consumes significant additional network resources compared to making a request using an existing connection.

How CICS Web support handles persistent connections

Persistent connections are the default behaviour for CICS Web support.

Before CICS TS 3.1, the connection behavior was that CICS would normally close the connection when data had been received from the Web client, unless the Web client sent a Connection: Keep-Alive header.

Now, when a connection is made between a Web client and CICS as an HTTP server, or between CICS as an HTTP client and a server, CICS attempts to keep the connection open by default.

When CICS is the HTTP server, the persistent connection is closed if:

- The user-written application that is handling the request closes the connection (by specifying the CLOSESTATUS(CLOSE) option on the WEB SEND command).
- The Web client closes the connection (notified by a Connection: close header).
- The Web client is an HTTP/1.0 client that does not send a Connection: Keep-Alive header.
- The timeout period is reached (indicating that the connection has failed, or that the Web client has deliberately exited the connection).

Otherwise, CICS leaves the persistent connection open for the Web client to send further requests. If there is a persistent connection with the client, CICS keeps the connection open after an error response is sent through a Web error program. The

exception is where CICS selects the 501 (Method Not Implemented) status code for the response, in which case the connection is closed by CICS.

With a TCPIPSERVICE resource definition for CICS Web support, the SOCKETCLOSE attribute of the TCPIPSERVICE definition should not be specified as zero. A zero setting for SOCKETCLOSE means that CICS as an HTTP server closes the connection immediately after receiving data from the Web client, unless further data is waiting. This means that persistent connections cannot be maintained.

When CICS is the HTTP client, the persistent connection is closed if:

*   The server closes the connection (notified by an HTTP/1.1 server sending a Connection: close header, or an HTTP/1.0 server failing to send a Connection: Keep-Alive header).
*   The user application program closes the connection (by specifying the CLOSESTATUS(CLOSE) option on the WEB SEND or WEB CONVERSE command, or by issuing a WEB CLOSE command).
*   End of task is reached and the connection has not yet been closed.

Virtual hosting

HTTP includes the concept of virtual hosting, where a single HTTP server can represent multiple hosts at the same IP address.

A DNS server can allocate several different host names to the same IP address. When an HTTP client makes a request to a particular host, it uses the DNS server to locate the IP address corresponding to that host name, and sends the request to that IP address.

In HTTP/1.0 the host name did not appear in the HTTP message; it was lost after the IP address had been resolved. This meant that if more than one set of resources was held on the server represented by the IP address, the server would have difficulty distinguishing which resources belonged to which host.

However, HTTP/1.1 requests provide the host name in the request (usually in a Host header). The presence of the host name in the message enables the HTTP server to direct requests containing different host names to the appropriate resources for each host. This feature of HTTP is known as virtual hosting. CICS Web support provides support for virtual hosting through the use of URIMAP definitions.

Administering virtual hosting

CICS supports virtual hosting through the URIMAP resource definition object.

Each URIMAP definition that you set up for CICS as an HTTP server (with USAGE(SERVER) in the URIMAP definition), includes the host name that the Web client is expected to supply in its request. CICS automatically creates virtual hosts for you, by grouping together into a single data structure all the URIMAP definitions in a CICS region that specify the same host name and the same TCPIPSERVICE definition. URIMAP definitions that specify no TCPIPSERVICE definition, and therefore apply to all of them, are added to all the data structures that specify a matching host name, so these URIMAP definitions might be part of more than one data structure. Each of these groups of URIMAP definitions then forms a virtual host that can be managed as a single unit.

You can use the following CICS commands to manage the virtual hosts that CICS has created from your URIMAP definitions:

- The INQUIRE HOST command is used to inquire on the status of a virtual host. The command tells you the host name of the virtual host, the TCPIPSERVICE definition with which it is associated (or if it is associated with every TCPIPSERVICE definition in the CICS region), and whether it is enabled or disabled.

- The **SET** HOST command is used to set the status of a virtual host to enabled or disabled. Disabling a virtual host means that all the URIMAP definitions that make up the virtual host cannot be accessed by applications. (However, note that a URIMAP definition that has been disabled in this way cannot be discarded.) When a virtual host is disabled, CICS returns an HTTP 503 response (Service Unavailable) to the Web client.

- The virtual host browsing commands are used to browse the virtual hosts in the CICS system.

The statistics program DFH0STAT includes a report showing the virtual hosts that CICS has created.

CICS automatically deletes virtual hosts if all the URIMAP definitions that made up the virtual host have been deleted. If you do not want to manage the virtual hosts that CICS has created for you, then you can ignore them, and manage at the level of your URIMAP definitions.

You can also process virtual hosts using an analyzer program. The host name for an HTTP request is passed to the analyzer program, and you can code the program to provide a host-dependent response to the request. However, virtual hosts that are set up in this way cannot be managed using the INQUIRE HOST, **SET** HOST and virtual host browsing commands.

Changes to CICS externals

Changes to resource definition

Changes to TCPIPSERVICE resource definition

The TCPIPSERVICE has a new attribute:

MAXDATALEN(32|*number*)
Defines the maximum length of data that can be received by CICS as an HTTP server, on the HTTP protocol or the USER protocol. The default value is 32K. The minimum is 3K, and the maximum is 524288K. To increase security for CICS Web support, specify this option on every TCPIPSERVICE definition for the HTTP protocol. It helps to guard against denial of service attacks involving the transmission of large amounts of data.

A new USER option is available on the PROTOCOL attribute. Processing for all non-HTTP requests must now be carried out under the USER protocol. No parsing is carried out for messages received on the USER protocol, and requests that have been divided up for transmission across the network are not automatically assembled. This is the same behavior as when handling non-HTTP messages in earlier CICS releases.

PROTOCOL(*ECI*|HTTP|IIOP|USER)
Specifies the application level protocol used on the TCP/IP port.

ECI The CICS ECI protocol is used.

HTTP HTTP protocol is used. HTTP protocol is handled by CICS Web support. CICS performs basic acceptance checks for messages sent and received using this protocol. This protocol is required for the well-known ports 80 (used for HTTP without SSL) and 443 (used for HTTP with SSL).

IIOP IIOP protocol is used. Specify IIOP for TCPIPSERVICEs that are to accept inbound requests for enterprise beans.

USER The user-defined protocol is used. Messages are processed as non-HTTP messages. They are flagged as non-HTTP and passed unchanged to the analyzer program for the TCPIPSERVICE. CICS Web support facilities are used for handling the request, but no acceptance checks are carried out for messages sent and received using this protocol. Processing for all non-HTTP requests must be carried out under the USER protocol, so that they are protected from the basic acceptance checks which CICS carries out for requests using the HTTP protocol. If an HTTP message is handled by the USER protocol, you are responsible for checking its validity.

The attributes of the TCPIPSERVICE resource definition that are used when PROTOCOL is set to USER, are the same as those used when PROTOCOL is set to HTTP. URIMAP definitions are not used with the USER protocol.

The new CICS-supplied transaction CWXU, the CICS Web user-defined protocol attach transaction, is the default when the protocol is defined as USER. CWXU executes program DFHWBXN.

The SOCKETCLOSE attribute is now described as follows. Note that SOCKETCLOSE should **not** be specified as 0 for the HTTP protocol.

SOCKETCLOSE(*NO*|*hhmmss*)
Specifies if, and for how long, CICS should wait before closing the socket, after issuing a receive for incoming data on that socket.

NO The socket is left open until it is closed by the client, or by a user application program in CICS.

hhmmss
The period of time (in HHMMSS format) after which CICS is to time out the socket. Choose a value that is appropriate to the responsiveness of the client, and the reliability of your network. Specifying 000000 closes the socket immediately if no data is available for any RECEIVEs other than the first one.

If you are using a TCPIPSERVICE for CICS Web Support with the HTTP protocol, SOCKETCLOSE(0) should not be specified. A zero setting for SOCKETCLOSE means that CICS closes the connection immediately after receiving data from the Web client, unless further data is waiting. This means that persistent connections cannot be maintained.

If you specify PROTOCOL(ECI) you must specify SOCKETCLOSE(NO).

The SOCKETCLOSE attribute does not apply to the first RECEIVE issued after a connection is made. On the first RECEIVE request, for the HTTP, USER and ECI protocols, CICS waits for data for 30 seconds before closing the socket. For the IIOP protocol, CICS waits indefinitely.

After the TCPIPSERVICE is installed, you cannot change this value using CEMT; you must set the TCPIPSERVICE out of service, then re-install the TCPIPSERVICE with the modified definition.

Changes to the application programming interface (HTTP/1.1 support)

New and changed commands

The following **EXEC CICS WEB** commands are enhanced when used by CICS as an HTTP server:

* **EXEC CICS WEB SEND**
* **EXEC CICS WEB RECEIVE**

CICS Web support is designed to allow Web-aware application programs that used these commands before CICS Transaction Server for z/OS, Version 3 Release 1 to work unchanged, until you choose to migrate them to take advantage of the enhancements that are now available.

The options that were available on the WEB SEND command before CICS Transaction Server for z/OS, Version 3 Release 1 are CLNTCODEPAGE, DOCTOKEN, LENGTH, STATUSCODE and STATUSTEXT. In CICS Transaction Server for z/OS, Version 3 Release 1:

* The name and function of the options DOCTOKEN, STATUSCODE and STATUSTEXT is unchanged.
* The options CLNTCODEPAGE and LENGTH are supported for migration purposes only, and their function is replaced by the new options CHARACTERSET and STATUSLEN respectively.
* Some new options are available for enhanced functionality.

The options that were available on the WEB RECEIVE command before CICS Transaction Server for z/OS, Version 3 Release 1 are CLNTCODEPAGE, HOSTCODEPAGE,INTO, LENGTH, MAXLENGTH, NOTRUNCATE, SET, and TYPE. In CICS Transaction Server for z/OS, Version 3 Release 1:

* The name and function of the options INTO, LENGTH, MAXLENGTH, NOTRUNCATE, SET, and TYPE is unchanged.
* The option HOSTCODEPAGE can still be used, but it is no longer required, and CICS can provide a default if it is not specified.
* The option CLNTCODEPAGE is supported for migration purposes only, and its function is replaced by the new option CHARACTERSET.
* Some new options are available for enhanced functionality.

The **EXEC CICS WEB SEND** and **WEB RECEIVE** commands also have a new range of options when used by CICS as an HTTP client, which is described in "Changes to the application programming interface (HTTP client requests)" on page 86.

The following EXEC CICS commands are provided or enhanced for both CICS as an HTTP server, and CICS as an HTTP client:

* **EXEC CICS WEB PARSE URL**
* **EXEC CICS WEB EXTRACT**
* **EXEC CICS CONVERTTIME**
* **EXEC CICS FORMATTIME**

These commands are described in "Changes to the application programming interface (General CICS Web support enhancements)" on page 146.

A new MAXDATALEN option is added to the EXTRACT TCPIP command:

MAXDATALEN(data-area)
> specifies a fullword binary field to contain the setting for the maximum length of data that can be received by CICS as an HTTP server.

WEB SEND (Server)

Send an HTTP response, or a non-HTTP message.

```
WEB SEND (CICS as an HTTP server)

►►──WEB──SEND────────────────────────────────────────────────────────►

  ►──┬─DOCTOKEN(data-value)──────────────────────────────┬────────────►
     └─FROM(data-area)──FROMLENGTH(data-value)──┬──────────────────────┘
                                                └─CHUNKING(cvda)─┘

  ►──┬──────────────────────┬──┬──────────────────┬───────────────────►
     └─MEDIATYPE(data-value)─┘  └─SERVERCONV(cvda)─┘

  ►──┬─CHARACTERSET(data-value)─┬──┬─HOSTCODEPAGE(data-value)─┬─────────►
     └─CLNTCODEPAGE(data-value)─┘  └──────────────────────────┘

  ►──┬──────────────────────┬─────────────────────────────────────────►
     └─STATUSCODE(data-value)─┘

  ►──┬─STATUSTEXT(data-area)──┬─STATUSLEN(data-value)─┬──┬─ACTION(cvda)─┬►
     └──────────────────────  └─LENGTH(data-value)────┘  └──────────────┘

  ►──┬─────────────────────┬──────────────────────────────────────────►◄
     └─CLOSESTATUS(cvda)───┘


Conditions: IOERR, INVREQ, NOTFND

This command is threadsafe.
```

Description

WEB SEND for CICS as an HTTP server selects an item for delivery by CICS Web support or the CICS business logic interface, and specifies options for sending it. The item can be:

- A response to an HTTP request that was made by a Web client, to CICS as an HTTP server.
- A non-HTTP message handled by CICS Web support facilities, with the user-defined (USER) protocol on the TCPIPSERVICE definition.
- A response to a request from another application that has used the CICS business logic interface to contact the program directly, rather than going through the CICS HTTP listener.

Only one response can be sent during a task. This can be a standard response using one WEB SEND command, or a chunked response using a sequence of WEB SEND commands.

Each time a request from a Web client is received, CICS starts a new task to process the request.

Options

ACTION*(cvda)*

specifies how the message should be sent out. The CVDA values that apply for CICS as an HTTP server are:

IMMEDIATE

sends the response immediately to the Web client. If CHUNKING is specified, the IMMEDIATE option is assumed. For message sends that do not use chunked transfer-coding, EVENTUAL is the default, which sends the response at end of task.

EVENTUAL

sends the response to the Web client at end of task. If CHUNKING is specified, the EVENTUAL option is ignored. This option produces the same behaviour as CICS Web support had in releases before CICS Transaction Server for z/OS, Version 3 Release 1, and it is the default for CICS as an HTTP server.

CHARACTERSET*(data-value)*

specifies a character set into which CICS translates the entity body of the item sent by the command before sending. The name of the character set can consist of up to 40 alphanumeric characters, including appropriate punctuation. CICS does not support all the character sets named by IANA.

When the CHARACTERSET option is specified, SERVERCONV(SRVCONVERT) is assumed, so code page conversion of the entity body takes place. As an alternative to selecting the character set yourself, specifying either SERVERCONV(SRVCONVERT), or HOSTCODEPAGE (if allowed), or both, and omitting CHARACTERSET, lets CICS determine a suitable character set for the message body.

If you omit all of the code page conversion options, no code page conversion takes place.

CHUNKING*(cvda)*

is used for controlling the message send when the message is being sent in chunks (known as chunked transfer-coding). The default when the option is not specified is that chunked transfer-coding is not in use. Chunked transfer-coding is only acceptable to HTTP/1.1 clients, and it cannot be used with HTTP/1.0 clients or non-HTTP messages.

The content of a chunked message can be divided into chunks in whatever way is most convenient for the application program. The body of a chunked message cannot be formed directly from CICS documents, so the DOCTOKEN option cannot be used.

Use a separate WEB SEND command with CHUNKING(CHUNKYES) for each chunk of the message. Use the FROM option to specify the chunk of data, and the FROMLENGTH option to specify the length of the chunk. Other options for the message, such as the CLOSESTATUS option, can be specified on the first

WEB SEND command of the sequence (which sends the first chunk), but do not specify them on subsequent commands (which send the second and subsequent chunks).

When you have sent the last chunk of the data, specify a further WEB SEND command with CHUNKING(CHUNKEND) and no FROM or FROMLENGTH option. CICS then sends an empty chunk to the recipient to complete the chunked message.

CVDA values are:

CHUNKNO
> Chunked transfer-coding is not used for the message. This is the default if the CHUNKING option is not specified.

CHUNKYES
> Chunked transfer-coding is in progress. The data specified by the FROM option represents a chunk of the message.

CHUNKEND
> Chunked transfer-coding is complete. No data is specified for this send. CICS sends an empty chunk to the recipient to complete the chunked message.

CLNTCODEPAGE(*data-value*)
This option is supported for migration purposes only. CHARACTERSET replaces it. The action taken by CICS is the same for both keywords. This means that code page conversion does take place when CLNTCODEPAGE is specified, even if the SERVERCONV option is not specified. Code page conversion does not take place if all the code page conversion options are omitted.

CLOSESTATUS(*cvda*)
specifies whether or not CICS closes the connection after sending the message. The default is that the connection is not closed. The CVDA values are:

CLOSE
> CICS writes a Connection header with the "close" connection option (Connection: close) for this response, and closes the connection with the Web client after sending the response. The header notifies the Web client of the closure. (For a Web client at HTTP/1.0 level, CICS achieves the same effect by omitting the Connection: Keep-Alive header.)
>
> If chunked transfer-coding is in use, the CLOSESTATUS(CLOSE) option can be specified on the first chunk of the message, to inform the Web client that the connection is closed after the chunked message is complete.

NOCLOSE
> means that the Connection: close header is not used for this response, and the connection is kept open. If the Web client is identified as HTTP/1.0 and has sent a Connection header with the "Keep-Alive" connection option (Connection: Keep-Alive), CICS sends the same header, to notify that a persistent connection will be maintained.

DOCTOKEN(*data-value*)
specifies the 16-byte binary token of a document to be sent as the message body. The document is created using the CICS Document interface (EXEC CICS DOCUMENT CREATE, INSERT, and SET commands), as described in

the *CICS Application Programming Guide*. The FROM option provides an alternative way to create a message body.

The body of a chunked message cannot be formed from CICS documents, so the DOCTOKEN option cannot be used for chunked transfer-coding.

CICS documents cannot be converted to the UTF-8 and UTF-16 character encodings.

FROM*(data-area)*
specifies a buffer of data which holds the complete message body, or a chunk of the message body. The message body is built by the application program. When you specify the FROM option, use the FROMLENGTH option to specify the length of the buffer of data. The DOCTOKEN option provides an alternative way to create the message body, but that option cannot be used for the body of a chunked message.

FROMLENGTH*(data-value)*
specifies the length, as a fullword binary value, of the buffer of data supplied on the FROM option. It is important to state this value correctly, because an incorrect data length can cause problems for the recipient of the message.

HOSTCODEPAGE*(data-value)*
specifies the 8-character name of the CICS (host) code page that was used by the application program for the entity body of the response. When the HOSTCODEPAGE option is specified, SERVERCONV(SRVCONVERT) is assumed, so code page conversion of the entity body takes place. Specifying either SERVERCONV(SRVCONVERT), or CHARACTERSET, or both, and omitting HOSTCODEPAGE, lets CICS identify the host code page.

If a CICS document is used to form the response body (DOCTOKEN option), do not specify the HOSTCODEPAGE option, because CICS identifies the host code page from the CICS document domain's record of the host code pages for the document.

If a buffer of data is used to form the response body (FROM option), you may need to specify HOSTCODEPAGE. The default if this option is not present is the default code page for the local CICS region, as set in the LOCALCCSID system initialization parameter. If you require code page conversion but your application has used a different code page, use HOSTCODEPAGE to specify it.

LENGTH*(data-value)*
This option is supported for migration purposes only. STATUSLEN replaces it.

MEDIATYPE*(data-value)*
specifies the data content of the message body, for example `text/xml`. The media type is up to 56 alphanumeric characters, including appropriate
\# punctuation, but not spaces.. CICS checks that the format of the media type is correct, but does not check the validity of the media type against the data content. CICS does not provide a default.

SERVERCONV*(cvda)*
specifies whether or not CICS translates the entity body of the item sent by the command before sending, from the code page used by the application, to a character set suitable for the recipient. You can use the CHARACTERSET and HOSTCODEPAGE options on this command to specify the character set and code page that are used. If you specify either of these options, code page conversion (SRVCONVERT) is assumed. Alternatively, you can omit either or both of these options, specify SERVERCONV(SRVCONVERT) and let CICS determine a suitable character set and code page.

SRVCONVERT

    CICS converts the entity body of the message.

NOSRVCONVERT

    CICS does not convert the entity body of the HTTP request, and it is sent to the server in the code page used by the application. If you specify NOSRVCONVERT, you cannot specify the CHARACTERSET or HOSTCODEPAGE options.

Note: If you omit all of the code page conversion options (SERVERCONV, CLNTCODEPAGE, CHARACTERSET, HOSTCODEPAGE), no code page conversion takes place.

STATUSCODE(*data-value*)

    specifies a standard HTTP status code determined by the application program, which is to be inserted on the status line of the HTTP response. The code is a halfword binary value. Examples are 200 (normal response) or 404 (not found). If this option is not specified, CICS supplies a default of 200.

    For status codes 204, 205, and 304, a message body is not allowed, and CICS returns an error response to the command if you attempt to include one. Other than that, CICS does not check that your use of the status code is appropriate.

STATUSLEN(*data-value*)

    specifies the length, as a fullword binary value, of the string supplied on the STATUSTEXT option.

STATUSTEXT(*data-area*)

    specifies a data-area containing human-readable text to describe the reason for the status code. The text is known as a reason phrase. Examples are "OK" (accompanying a 200 status code), or "Bad Request" (accompanying a 400 status code).

WEB RECEIVE (Server)

Receive an HTTP request, or a non-HTTP message.

WEB RECEIVE (CICS as an HTTP server)

```
►►──WEB──RECEIVE──┬─INTO(data-area)─┬──LENGTH(data-area)──────────────►
                  └─SET(ptr-ref)────┘

►──┬──────────────────────┬──┬────────────┬──┬───────────┬────────────►
   └─MAXLENGTH(data-value)─┘  └─NOTRUNCATE─┘  └─TYPE(cvda)─┘

►──┬──────────────────┬──┬─CHARACTERSET(data-value)─┬─────────────────►
   └─SERVERCONV(cvda)──┘  └─CLNTCODEPAGE(data-value)─┘

►──┬────────────────────────┬────────────────────────────────────────►◄
   └─HOSTCODEPAGE(data-value)─┘
```

Conditions: INVREQ, LENGERR, NOTFND

This command is threadsafe.
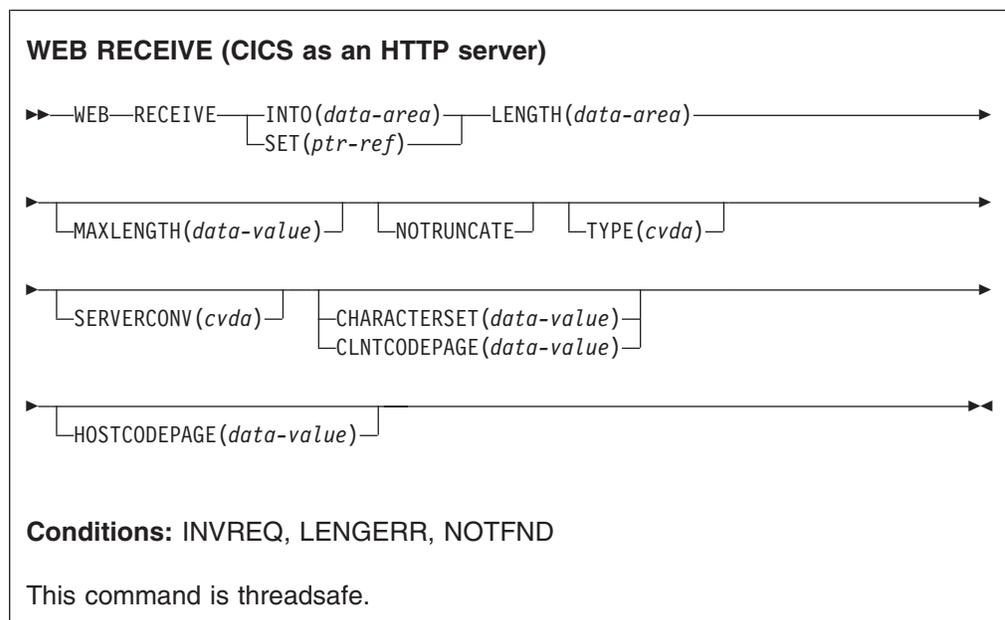
Description

WEB RECEIVE receives the body of an HTTP request, or all the data for a non-HTTP message, into an application-supplied buffer. The headers for an HTTP request can be examined separately using the WEB HTTPHEADER commands. The item received by the WEB RECEIVE command can be:

- The body of an HTTP request that a Web client has made to CICS as an HTTP server.
- A non-HTTP message handled by CICS Web support facilities, with the user-defined (USER) protocol on the TCPIPSERVICE definition.
- A request from another application that has used the CICS business logic interface to contact the application program directly, rather than going through the CICS HTTP listener.

The data is returned in its escaped form. The type of code page conversion used for incoming data received by the CICS application program can be specified on this command. If you omit all of the code page conversion options (SERVERCONV, CLNTCODEPAGE, CHARACTERSET, HOSTCODEPAGE), no code page conversion takes place.

Options

CHARACTERSET(*data-value*)
specifies the character set that was used by the Web client for the entity body of the received item. The name of the character set can consist of up to 40 alphanumeric characters, including appropriate punctuation. CICS does not support all the character sets named by IANA.

When the CHARACTERSET option is specified, SERVERCONV(SRVCONVERT) is assumed, so code page conversion of the entity body takes place. As an alternative to identifying the character set yourself, specifying either SERVERCONV(SRVCONVERT), or HOSTCODEPAGE, or both, and omitting CHARACTERSET, lets CICS identify the character set for the message body.

CLNTCODEPAGE(*data-value*)
This option is supported for migration purposes only. CHARACTERSET replaces it. The action taken by CICS is the same for both keywords. This means that code page conversion does take place when CLNTCODEPAGE or HOSTCODEPAGE is specified, even if the SERVERCONV option is not specified.

HOSTCODEPAGE(*data-value*)
specifies the 8-character name of the CICS (host) code page used by the application program, into which the entity body of the received item should be converted from the character set in which it was received from the Web client. When the HOSTCODEPAGE option is specified, SERVERCONV(SRVCONVERT) is assumed, so code page conversion of the entity body takes place. Specifying either SERVERCONV(SRVCONVERT), or CHARACTERSET, or both, and omitting HOSTCODEPAGE, lets CICS determine the host code page.

The default if this option is not specified is the default code page for the local CICS region, as specified in the LOCALCCSID system initialization parameter.

INTO(*data-area*)
specifies the buffer that is to contain the data being received.

LENGTH*(data-area)*

specifies a fullword binary variable which is set to the amount of data that CICS has returned to the application.

- If the NOTRUNCATE option **is not** specified, any further data present in the message has now been discarded. A LENGERR response with a RESP2 value of 57 is returned if further data was present.

- If the NOTRUNCATE option **is** specified, any additional data is retained. A LENGERR response with a RESP2 value of 36 is returned if additional data is available. The description for the NOTRUNCATE option tells you what to do in this case.

MAXLENGTH*(data-value)*

specifies the maximum amount, as a fullword binary value, of data that CICS is to pass to the application. The MAXLENGTH option applies whether the INTO or the SET option is specified for receiving data. If the data has been sent using chunked transfer-coding, CICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the chunked message, rather than to each individual chunk. The data is measured after any code page conversion has taken place.

NOTRUNCATE

specifies that when the data available exceeds the length requested on the MAXLENGTH option, the remaining data is not to be discarded immediately but is to be retained for retrieval by subsequent RECEIVE commands. (If no further RECEIVE commands are issued, the data is discarded during transaction termination.)

A single RECEIVE command using the SET option and without the MAXLENGTH option receives all the remaining data, whatever its length. Alternatively, you can use a series of RECEIVE commands with the NOTRUNCATE option to receive the remaining data in appropriate chunks. Keep issuing the RECEIVE command until you are no longer getting a LENGERR response.

SERVERCONV*(cvda)*

specifies whether or not CICS translates the entity body of the item received, from the character set used by the Web client, to a code page suitable for the application. You can use the CHARACTERSET and HOSTCODEPAGE options on this command to specify the character set and code page that are used. If you specify either of these options, code page conversion (SRVCONVERT) is assumed. Alternatively, you can omit either or both of these options, specify SERVERCONV(SRVCONVERT) and let CICS determine a suitable character set and code page.

SRVCONVERT

CICS converts the entity body of the message.

NOSRVCONVERT

CICS does not convert the entity body of the item, and it is passed to the application in the character set used by the Web client. If you specify NOSRVCONVERT, you cannot specify the CHARACTERSET or HOSTCODEPAGE options.

SET*(ptr-ref)*

specifies a pointer reference that is to be set to the address of data received. The pointer reference is valid until the next receive command or the end of task.

TYPE(*cvda*)
returns the type of request received. CVDA values are:

HTTPYES
indicates an HTTP request.

HTTPNO
indicates a non-HTTP request.

\# In CICS Transaction Server for z/OS, Version 3, HTTP requests and non-HTTP
\# requests use different protocols, which are specified on TCPIPSERVICE
\# definitions, and must therefore use different ports. Non-HTTP requests use the
\# user-defined (USER) protocol. You might use the TYPE option to distinguish
\# between the request types if you specify the same user-written application
\# program for responding to both HTTP and non-HTTP requests.

Changes to the system programming interface

The MAXDATALEN option is added to the **CREATE TCPIPSERVICE**, **INQUIRE TCPIPSERVICE** and **SET TCPIPSERVICE** commands:

MAXDATALEN(32│*number*)
specifies the maximum length of data, in kilobytes, that may be received by CICS as an HTTP server. The default value is 32. The minimum is 3, and the maximum is 524288.

Also on the CREATE TCPIPSERVICE and INQUIRE TCPIPSERVICE commands, the new option USER, meaning the user-defined protocol, is available on the PROTOCOL attribute.

The new CICS-supplied transaction CWXU, the CICS Web user-defined protocol attach transaction, is the transaction to be specified when the protocol is defined as USER. CWXU executes program DFHWBXN.

On the CREATE TCPIPSERVICE command, note that for the HTTP protocol, 0 should not be specified for the SOCKETCLOSE option, because this setting means that persistent connections cannot be maintained.

The following commands are provided to manage virtual hosts:

* **INQUIRE HOST**
* **SET HOST**

CICS automatically manages the creation and deletion of virtual hosts, based on the URIMAP definitions in your CICS region.

INQUIRE HOST
Retrieve information about virtual hosts in the local system.

```
INQUIRE HOST

►►──INQUIRE HOST(data-area)───────────────────────────────────────►
                            └─ENABLESTATUS(cvda)─┘

►──────────────────────────────────────────────────────────────────►◄
   └─TCPIPSERVICE(data-area)─┘
```

Conditions: END, ILLOGIC, INVREQ, NOTAUTH, NOTFND

This command is threadsafe.
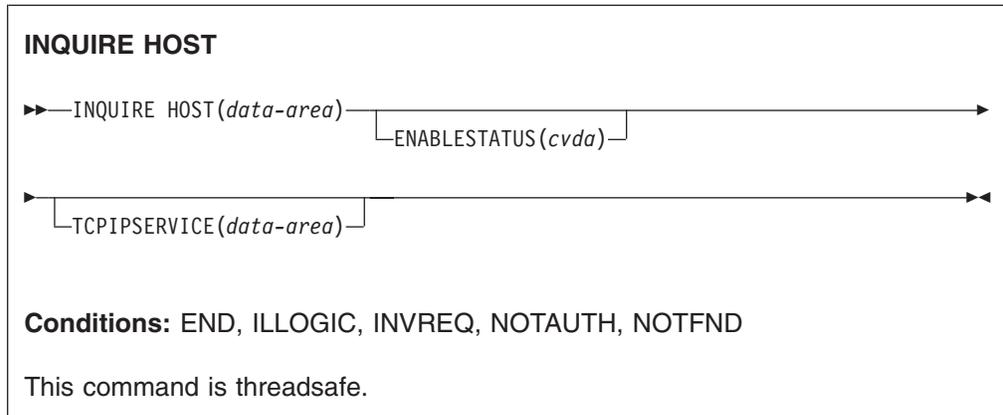
Description

The INQUIRE HOST command allows you to retrieve information about a particular virtual host in the local CICS region. Virtual hosting in the *CICS Internet Guide* explains what virtual hosts are. Virtual hosts are based on the URIMAP resource definition object. CICS automatically creates virtual hosts for you, by grouping together into a single data structure all the URIMAP definitions in a CICS region that specify the same host name and the same TCPIPSERVICE. URIMAP definitions that specify no TCPIPSERVICE, and therefore apply to all of them, are added to all the data structures that specify a matching host name, so these URIMAP definitions might be part of more than one data structure.

Browsing

You can also browse through all the virtual hosts that exist in the region, using the browse options (START, NEXT, and END) on INQUIRE HOST commands. See Browsing resource definitions for general information about browsing, including syntax, exception conditions, and examples.

Options

HOST(*data-value*)
    specifies the name of a virtual host. The name of each virtual host is taken from the host name specified in the URIMAP definitions that make up the virtual host. For example, if your CICS region contained URIMAP definitions that specified a host name of www.example.com, CICS would create a virtual host with the name www.example.com. A host name in a URIMAP definition can be up to 120 characters.

ENABLESTATUS(*cvda*)
    returns a CVDA value indicating the status of this virtual host. CVDA values are:

    ENABLED
            The virtual host is enabled.

    DISABLED
            The virtual host is disabled. The URIMAP definitions that make up the virtual host cannot be accessed by applications.

TCPIPSERVICE(*data-area*)
    returns the 1- to 8-character name of the TCPIPSERVICE definition that specifies the inbound port to which this virtual host relates. If this definition is not given, the virtual host relates to all TCPIPSERVICE definitions.

SET HOST

Sets the status of a virtual host to enabled or disabled.

```
SET HOST

►►──SET HOST(data-area)─────────────────────────────────────►◄
                      └─ENABLESTATUS(cvda)─┘


Conditions: NOTAUTH, NOTFND

This command is threadsafe.
```
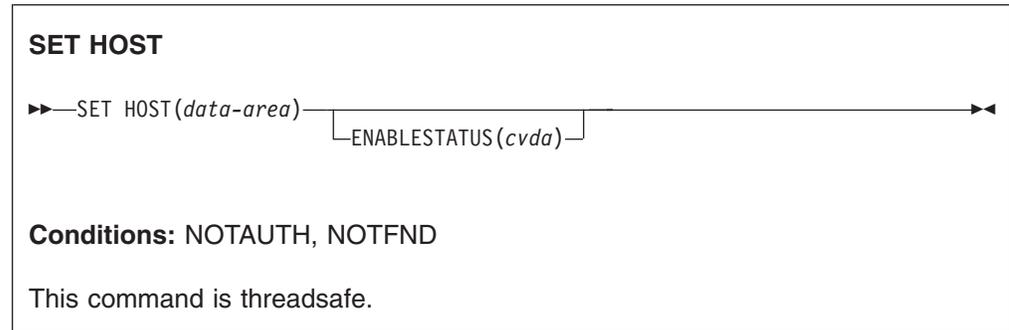
Description

The SET HOST command is used to set the status of a virtual host to enabled or
disabled. Disabling a virtual host means that all the URIMAP definitions that make
up the virtual host cannot be accessed by applications. When a virtual host is
disabled, CICS returns a HTTP response with a 503 (Service Unavailable) status
code to Web clients.

When the INQUIRE URIMAP command is used to inquire on an individual URIMAP
definition, a special status DISABLEDHOST is returned to indicate that the virtual
host is disabled. You do not need to change the disabled status of the URIMAP
definitions individually; the SET HOST command can be used to re-enable all the
URIMAP definitions that make up the virtual host. However, note that a URIMAP
definition with the DISABLEDHOST status cannot be discarded. If you want to
discard the definition, it must be disabled individually (using the SET URIMAP
command).

Options

HOST(*data-area*)
> specifies the name of a virtual host. The name of each virtual host is taken from
> the host name specified in the URIMAP definitions that make up the virtual
> host. For example, if your CICS region contained URIMAP definitions that
> specified a host name of www.example.com, CICS would create a virtual host
> with the name www.example.com. A host name in a URIMAP definition can be
> up to 120 characters.

ENABLESTATUS(*cvda*)
> CVDA values are:
>
> **ENABLED**
>> The URIMAP definitions that make up the virtual host can be accessed
>> by applications.
>
> **DISABLED**
>> The URIMAP definitions that make up the virtual host cannot be
>> accessed by applications.

Changes to CEMT

The **INQUIRE HOST** and **SET HOST** commands are added to the CEMT transaction.
CICS Supplied Transactions has information about these commands.

Changes to statistics

A new statistic Maxdata (field name SOR_MAXDATA_LENGTH) is added to the TCP/IP Services resource statistics. This statistic shows the MAXDATALEN settings for the TCPIPSERVICE definitions. TCP/IP Services resource statistics are collected by the COLLECT STATISTICS command using the TCPIPSERVICE keyword, and are mapped by the DFHSORDS DSECT.

The statistics program DFH0STAT includes a new report showing the virtual hosts that CICS has created from your URIMAP definitions. The Virtual Hosts report displays the name of each virtual host, whether it is enabled or disabled, and the name of the TCPIPSERVICE definition with which it is associated. If no TCPIPSERVICE definition is shown, the virtual host is associated with every TCPIPSERVICE definition in the CICS region.

Chapter 5. General enhancements to CICS Web support

Enhancements to CICS Web support include improved capability for processing HTTP requests and responses when CICS is an HTTP server, improvements to the way CICS Web support handles code page conversion, and better support for HTTP time and date formats.

Improved support for processing HTTP requests and responses

The URIMAP definition is a new CICS resource which provides an improved and more powerful facility for processing HTTP requests and responses when CICS is an HTTP server. The new URIMAP resource definition means that:

- As well as providing dynamic responses to a Web client using an application program, you can provide static responses using content from an HFS file or CICS document template.
- You can handle HTTP requests with greater transparency.
- You can carry out online administration for HTTP requests.
- The use of an analyzer program to handle HTTP requests is optional.

CICS API support for analyzing HTTP requests and responses is extended. The `EXEC CICS WEB EXTRACT` command is enhanced to extract more information from requests, and the new `EXEC CICS WEB PARSE URL` command enables you to analyze any URL and extract information for reuse.

The `EXEC CICS WEB SEND` and WEB RECEIVE commands used by CICS as an HTTP server are enhanced to give you more control and capability when receiving requests from a Web client and sending responses. For example, you can now use a buffer of data to provide the message body for a response (as an alternative to using a CICS document), and you can specify options to determine when the response is sent and whether CICS should signal the client to end its connection after receiving the response. The changes to these commands are described in Chapter 4, "CICS Web support upgrade to HTTP/1.1," on page 111.

Improved support for code pages

The code page conversion for CICS Web support is improved, and you no longer need to set up a code page conversion table (DFHCNV) for use with CICS Web support.

The UTF-8 and UTF-16 character encodings are now available for code page conversion in CICS.

Support for HTTP time and date format

A new CICS API command, CONVERTTIME, converts common date and time stamp formats used on the Internet into the CICS ABSTIME format. Options are added to the `EXEC CICS FORMATTIME` command to convert the ABSTIME format into a date and time stamp string that is suitable for use on the Internet.

Benefits of CICS Web support enhancements

The enhancements to CICS Web support provide the following benefits:

- CICS as an HTTP server can be managed more easily by using URIMAP resource definitions instead of, or as well as, an analyzer program. URIMAP definitions can be used to assign an HTTP request directly to a user application program, and you can administer them online.
- Using URIMAP definitions, you can make CICS automatically provide static responses to a Web client, so you do not need to write and invoke an application program to provide this type of response. The static response can be formed from a CICS document template or a z/OS UNIX System Services HFS file, and the response can incorporate data from a Web client's query string.
- The UTF-8 and UTF-16 character encodings are available for code page conversion. The code page conversion process for CICS Web support is improved, and you no longer need to set up a code page conversion table (DFHCNV) for use with CICS Web support.
- The CICS API supports translation between date and time stamp formats used on the Internet, and the CICS ABSTIME format.

Terminology

The following terminology is used in connection with the enhancements to CICS Web support:

CICS as an HTTP server
> The process where CICS receives HTTP requests from Web clients, and sends responses. A user application program may be used to process the request and provide the response, or a static response may be specified using a URIMAP definition.

CICS as an HTTP client
> The process where a user application program sends requests through CICS to HTTP servers, and receives responses.

Static response
> An HTTP response that is constructed by CICS from a document template or HFS file specified by a URIMAP definition.

Application-generated response
> An HTTP response that is built dynamically by a user application program. This can be either a Web-aware application program or a non-Web-aware application program.

Web-aware application program
> An application program that uses the `EXEC CICS WEB` application programming interface commands to receive a Web client's request and send an HTTP response.

Non-Web-aware application program
> For CICS Web support, an application program that does not use the `EXEC CICS WEB` application programming interface commands. These programs can be enabled for the Web using a converter program, which translates the Web client's request into acceptable input, and composes an HTTP response based on the program's output.

Web client
> Any client application that makes an HTTP request to CICS as an HTTP server. This might be a Web browser that displays responses to a human user, or an automatic user agent (such as an information gatherer for a search engine), or an application program (such as a CICS application that makes HTTP client requests).

Chunked transfer-coding (also known as chunking)
The process where the body of an HTTP message is transferred as a series of chunks, each with its own chunk size header.

Pipelining
The process where a client sends multiple HTTP requests to a server without waiting for a response. Responses must then be returned from the server in the same sequence that the requests were received.

Idempotency
A property of an individual HTTP method or a pipelined sequence of requests. If a method is idempotent, the same result is always obtained when you repeat the same request with that method. If a request sequence is idempotent, the same result is always obtained when all, or part, of the series of requests is repeated.

Persistent connection
A connection between a Web client and an HTTP server which can be reused for more than one exchange of a request and a response.

URL   A URL (Uniform Resource Locator) is a specific type of URI (Universal Resource Identifier). A URI can name any resource, whereas a URL normally locates an existing resource on the Internet.

Virtual hosting
The situation where a single HTTP server can represent multiple hosts at the same IP address. Each host name that is provided in this way is known as a virtual host.

Requirements

There are no special hardware or software requirements to support this function.

Related information:

HTTP request and response processing for CICS as an HTTP server

HTTP requests for CICS as an HTTP server are initiated by a Web client that makes a request to CICS. CICS provides the Web client with responses to the requests it makes. The responses can be created from a static document identified by a URIMAP resource definition, or they can be created dynamically by a user application program.

This figure contains a high-resolution graphic that is not supported in this display format. To view the graphic, please use the CICS Information Center.

Processing for CICS as an HTTP server takes place as follows:

1. **CICS receives a TCP/IP connection request.** The CICS Sockets domain uses the TCPIPSERVICE resource definition for the port to determine that the request should be processed by CICS Web support.
2. **CICS matches the URL for the request to a URIMAP definition, if available.** If a successful match is made, the URIMAP definition tells CICS how to process the request.
3. **If the URIMAP definition specifies redirection, CICS redirects the Web client to the specified URL.** CICS composes the redirection message and transmits it to the Web client.

4. **If the URIMAP definition specifies a static response, CICS forms and supplies the response.** CICS uses a document template or a z/OS UNIX System Services HFS file, together with appropriate HTTP headers, to form an HTTP response. The response undergoes appropriate code page conversion, and CICS then transmits the response to the Web client.

5. **An analyzer program may be run, if the URIMAP definition specifies its use, or if no matching URIMAP definition is found.** The analyzer program can interpret the request dynamically, or it can be used for monitoring or audit purposes.

6. **An application program is executed to service the request.** You can specify the application program using a URIMAP definition, or using an analyzer program. A Web-aware application program, using the `EXEC CICS` WEB and `EXEC CICS` DOCUMENT application programming interfaces, can be used to handle the request and construct a response. A non-Web-aware application program can be enabled for the Web using either a converter program (which translates the Web client's request into acceptable input, and composes an HTTP response based on the program's output), or a Web-aware application program that calls the non-Web aware program and uses its output.

7. **CICS generates some required HTTP headers and adds them to the message.** Appropriate headers are generated depending on the HTTP version for the response.

8. **CICS transmits the complete HTTP response to the Web client.** If the Web client supports persistent connections, CICS keeps the connection open for further possible HTTP requests, until the user application or Web client requests closure or the timeout period is reached.

During this process, code page conversion is usually needed when messages enter and leave the CICS environment, so that CICS Web support processing and user-written applications (which typically use an EBCDIC encoding) can communicate with Web clients (which typically use an ASCII encoding).

Unicode UTF-8 and UTF-16 code page conversion in CICS Web support

CICS Web support can now handle code page conversion to, from and between the Unicode UTF-8 and UTF-16 character encodings.

Code page conversion for UTF-8 and UTF-16 uses conversion services provided by z/OS. The conversion facility must be enabled in your z/OS system, as described in *z/OS Support for Unicode: Using Conversion Services*.

Code page conversion is usually needed so that CICS Web support processing and user-written applications, which typically use an EBCDIC encoding, can communicate with Web clients, which typically use an ASCII encoding. Inbound messages are converted to an EBCDIC encoding for the application to process, and outbound messages generated by the application are converted from an EBCDIC encoding into a suitable character set for the Web client.

The UTF-8 and UTF-16 character encodings can be used for the message body of an item that CICS receives from, or sends to, the Internet. You can convert to and from the UTF-8 and UTF-16 character encodings with any of the EBCDIC code pages that CICS supports.

Note that CICS documents and document templates cannot be converted to the UTF-8 and UTF-16 character encodings. If you want to send an outbound message in these character encodings, use the FROM option on the WEB SEND or WEB CONVERSE command to specify a buffer of data to form the message body, rather than using the DOCTOKEN option to specify a CICS document.

Handling HTTP date and time stamp formats

An application program that interacts with the Web through CICS might need to produce the correct date and time stamp format in HTTP headers. Application programs might also need to work with date and time stamp information received from the Web. The CONVERTTIME and FORMATTIME commands enable you to handle common architected date and time stamp string formats used for HTTP.

You can use the CONVERTTIME command to convert an architected date and time stamp string to ABSTIME. You do not need to identify the format of the date and time stamp; the CONVERTTIME command recognizes and converts three different date and time stamp formats which are commonly used on the Internet.

You can use the FORMATTIME command to convert the current date and time (in ABSTIME format), or a date and time produced by the application program (such as an expiry date), into the RFC 1123 format.

Changes to CICS externals

Changes to resource definition

New attribute HFSFILE on DOCTEMPLATE resource definition

This attribute enables a z/OS UNIX System Services HFS file to be used as a document template.

HFSFILE(_hfsfile_**)**
When the template resides in a z/OS UNIX System Services HFS file, this specifies the fully qualified (absolute) or relative name of the HFS file. The name can be specified as an absolute name including all directories and beginning with a slash, for example, `/u/facts/images/bluefish.jpg`. Alternatively, it can be specified as a name relative to the HOME directory of the CICS region userid, for example, `facts/images/bluefish.jpg`. Up to 255 characters can be used. The CICS region must have permissions to access z/OS UNIX, and it must have permission to access the HFS directory containing the file, and the file itself.

URIMAP resource definitions
URIMAP definitions are resource definitions that match the URIs of HTTP or Web service requests, and provide information on how to process the requests.

URIMAP definitions are used to provide three different Web-related facilities in CICS:
1. **Requests from a Web client, to CICS as an HTTP server.** URIMAP definitions for requests for CICS as an HTTP server have a USAGE attribute of SERVER. These URIMAP definitions match the URLs of HTTP requests that CICS expects to receive from a Web client, and they define how CICS should provide a response to each request. You can use a URIMAP definition to tell CICS to:
   - **Provide a static response** to the HTTP request, using a document template or z/OS UNIX System Services HFS file.

- **Provide a dynamic response** to the HTTP request, using an application program.
2. **Requests to a server, from CICS as an HTTP client.** URIMAP definitions for requests from CICS as an HTTP client have a USAGE attribute of CLIENT. These URIMAP definitions specify URLs that are used when a user application, acting as a Web client, makes a request through CICS Web support to an HTTP server. Setting up a URIMAP definition for this purpose means that you can avoid identifying the URL in your application program.
3. **Web service requests.** URIMAP definitions for Web service requests have a USAGE attribute of PIPELINE. These URIMAP definitions associate a URI for an inbound Web service request (that is, a request by which a client invokes a Web service in CICS) with a PIPELINE or WEBSERVICE resource that specifies the processing to be performed.

For CICS as an HTTP server, URIMAP definitions incorporate most of the functions that were previously provided by the analyzer program associated with the TCPIPSERVICE definition. An analyzer program may still be involved in the processing path if required.

URIMAP definition attributes:

```
►►──URIMAP(name)──GROUP(groupname)──┬──────────────────┬──────────────────►
                                    └─DESCRIPTION(text)─┘


   ┌─STATUS(ENABLED)──┐                    ┌─SCHEME(HTTP)──┐
►──┤                  ├──PATH(path)──┬─────┴───────────────┴──────────────►
   └─STATUS(DISABLED)─┘              └─SCHEME(HTTPS)─┘


   ┌─USAGE(SERVER)──┤ SERVER attributes ├──────┐
►──┤                                           ├────────────────────────►◄
   ├─USAGE(CLIENT)──┤ CLIENT attributes ├──────┤
   └─USAGE(PIPELINE)─┤ PIPELINE attributes ├───┘
```

SERVER attributes:

```
#  ├──┬─HOST(hostname)─┬──┬──────────────────────┬──────────────────────►
      └─HOST(*)────────┘  └─TCPIPSERVICE(name)───┘


   ├──┬──────────────────────────────────────────────────────────────────
      └─MEDIATYPE(type)──CHARACTERSET(characterset)──HOSTCODEPAGE(codepage)──
      ┌─ANALYZER(YES)─┐
      └─ANALYZER(NO)──┘  └─CONVERTER(name)─┘  └─TRANSACTION(name)─┘  └─PROGRAM(name)─┘  └─USERID(id)─┘

   ┌─TEMPLATENAME(name)─┐
   ┴─────────────────────┴──
   └─HFSFILE(name)──────┘


   ├──┬─REDIRECTTYPE(NONE)──────┬──┬────────────────┬──────────────────────┤
      ├─REDIRECTTYPE(TEMPORARY)─┤  └─LOCATION(url)──┘
      └─REDIRECTTYPE(PERMANENT)─┘
```

CLIENT attributes:

```
#  ├──HOST(hostname)──┬────────────────────┬──┬─────────────────┬────────┤
                      └─CERTIFICATE(label)─┘  └─CIPHERS(value)──┘
```

PIPELINE attributes:

```
#  ├──┬─HOST(hostname)─┬──PIPELINE(name)──┬───────────────────┬───────────►
      └─HOST(*)────────┘                  └─WEBSERVICE(name)──┘


►──┬─────────────────────┬──┬──────────────────┬──┬────────────┬──────────►
   └─TCPIPSERVICE(name)──┘  └─TRANSACTION(name)─┘  └─USERID(id)─┘


►──┬─REDIRECTTYPE(NONE)──────┬──┬────────────────┬────────────────────────┤
   ├─REDIRECTTYPE(TEMPORARY)─┤  └─LOCATION(url)──┘
   └─REDIRECTTYPE(PERMANENT)─┘
```

ANALYZER({NO|YES})

> **This attribute is for USAGE(SERVER), where an application-generated response is to be provided.** (For USAGE(CLIENT) or USAGE(PIPELINE), the attribute is forced to NO.) It specifies whether an analyzer program is to be used in processing the HTTP request. The analyzer that can be run using this attribute is the analyzer associated with the TCPIPSERVICE definition or definitions to which this URIMAP definition relates. (An analyzer program must be in the local CICS region.) YES runs the analyzer. NO means that the analyzer is not used.

> **Acceptable characters:**
>
> ```
> A-Z 0-9 $ @ #
> ```
>
> Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase.

CERTIFICATE(*label*)

> **This attribute is for USAGE(CLIENT).** It specifies the label of the X.509 certificate that is to be used as the SSL client certificate during the SSL handshake. Certificate labels can be up to 32 characters long. This attribute is only used when the URI specified by the URIMAP definition is to be used for an HTTPS request made by CICS as a client. It is up to the server to request a SSL client certificate, and if this happens, CICS supplies the certificate label which is specified in the URIMAP definition. If this attribute is omitted, the default certificate defined in the key ring for the CICS region user ID is used. The certificate must be stored in a key ring in the external security manager's database.

CIPHERS(*value*)

> **This attribute is for USAGE(CLIENT).**
>
> Specifies a string of up to 56 hexadecimal digits that is interpreted as a list of up to 28 2-digit cipher suite codes. When you use the CEDA transaction to define the resource, CICS automatically initializes the attribute with a default list of acceptable codes. For CICS to initialize the attribute, the KEYRING system initialization parameter must be specified in the CICS region where you are running CEDA. If KEYRING is not set, CICS does not initialize the attribute. The default list of codes depends on the level of encryption that is specified by the ENCRYPTION system initialization parameter.
>
> - For ENCRYPTION=WEAK, the default value is `03060102`
> - For ENCRYPTION=MEDIUM, the initial value is `0903060102`
> - For ENCRYPTION=STRONG, the initial value is `0504352F0A0903060102`
>
> You can reorder the cipher codes or remove them from the initial list. However, you cannot add cipher codes that are not in the default list for the specified encryption level. To reset the value to the default list of codes, delete all of the cipher suite codes and the field will automatically repopulate with the default list.

CHARACTERSET(*characterset*)

> **This attribute is for USAGE(SERVER), where a static response is to be provided.** It specifies the 1-40 character name of the character set into which CICS converts the entity body of the response that is sent to the Web client. CICS does not support all the character sets named by IANA. The value of this attribute is included in the Content-Type header of the response.
>
> CHARACTERSET must be specified if a static response is being provided and the MEDIATYPE attribute specifies a text type.

CONVERTER(*name*)

> **This attribute is for USAGE(SERVER), where an application-generated response is to be provided.** It specifies the 1-8 character name of a converter program that is to be run to perform conversion or other processing on the request and response. Typically, a converter program transforms the HTTP request into a COMMAREA that can be used by an application program, and transforms the output into an HTTP response. The converter program can be any converter program that is available in the local CICS region.

> **Acceptable characters:**
>
> A-Z 0-9 $ @ #
>
> Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase.

DESCRIPTION(*text*)

You can provide a description of the resource you are defining in this field. The description text can be up to 58 characters in length. There are no restrictions on the characters that you can use. However, if you use parentheses, ensure that for each left parenthesis there is a matching right one. If you use the CREATE command, for each single apostrophe in the text, code two apostrophes.

GROUP(*groupname*)

Every resource definition must have a GROUP name. The resource definition becomes a member of the group and is installed in the CICS system when the group is installed.

> **Acceptable characters:**
>
> A-Z 0-9 $ @ #
>
> Any lower case characters you enter are converted to upper case.

The GROUP name can be up to eight characters in length. Lowercase characters are treated as uppercase characters. Do not use group names beginning with DFH, because these characters are reserved for use by CICS.

HFSFILE(*name*)

This attribute is for USAGE(SERVER), where a static response is to be provided. It specifies the fully qualified (absolute) or relative name of a z/OS UNIX System Services HFS file that forms the body of the static response which is sent to the HTTP request from the Web client. The name can be specified as an absolute path including all directories and beginning with a slash, for example, `/u/facts/images/bluefish.jpg`. Alternatively, it can be specified as a path relative to the HOME directory of the CICS region userid, for example, `facts/images/bluefish.jpg`. Up to 255 characters can be used.

> **Acceptable characters:**
>
> A-Z a-z 0-9 . / _ # @

The value specified must be a valid name for an HFS file:
* The name must not contain imbedded space characters
* The name must not contain consecutive instances of the / character

The name is case-sensitive.

Note: Resource level security is not applied to items delivered as a static response. If you need to apply access controls based on a user ID to an item delivered in this way, you need to deliver the material as an application-generated response instead.

If you want to use path matching, include an asterisk as a wildcard character at the end of the path for the HFS file, and also at the end of the path specified by the PATH attribute. CICS takes the portion of each HTTP request's path that is covered by the wildcard character, and substitutes this as the last part of the file path.

For example, you could create a URIMAP definition with the PATH attribute specified as:

```
findout/pictures/*
```

and the HFSFILE attribute specified as:

```
/u/facts/images/*
```

The URIMAP definition is used to process an incoming HTTP request

```
http://www.example.com/findout/pictures/bluefish.jpg
```

CICS appends `bluefish.jpg` to the HFS file path specified in the URIMAP definition in place of the asterisk, so that the HFS file

```
/u/facts/images/bluefish.jpg
```

is used as the static response.

> **Note:** You cannot use an asterisk alone in the HFSFILE specification. At least one level of the directory structure must be specified.

A query string cannot be substituted into an HFS file (unless you define the HFS file as a CICS document template, and specify it using the TEMPLATENAME option instead of the HFSFILE option).

HOST(*hostname*|*)
This attribute is for all USAGE options. It specifies the host component of the URI to which the URIMAP definition applies, which can be up to 116 characters. An example of a host name is `www.example.com`.

The HOST attribute must be present, and must contain only alphanumeric characters, hyphens (-) or periods (.). Hexadecimal escape sequences cannot be used in a host name. CICS validates this at define time. The host name can be entered in any case, but it is converted to lower case in the URIMAP definition.

An IPv4 address can be used as a host name, but IPv6 addresses are not supported.

When USAGE(SERVER) or USAGE(PIPELINE) is specified, a single asterisk can be used as the HOST attribute. This makes the URIMAP definition match any host name. An asterisk cannot be used as a wildcard in the HOST attribute along with any other characters.

For URIMAP definitions relating to CICS as an HTTP client, USAGE(CLIENT), if you need to specify a port number in the URL for the request to the server, include it in the HOST attribute, together with the colon preceding it. You only need to specify the port number if it is other than the default for the scheme (80 for HTTP without SSL, or 443 for HTTPS, HTTP with SSL).

HOSTCODEPAGE(*codepage*)
This attribute is for USAGE(SERVER), where a static response is to be provided. It specifies the 1-10 character name of the IBM code page (EBCDIC) in which the text document that forms the static response is encoded. This information is needed by CICS to perform code page conversion for the entity body of the static response.

HOSTCODEPAGE must be specified if a static response is being provided and the MEDIATYPE attribute specifies a text type.

LOCATION(*url*)
This attribute is for USAGE(SERVER) and USAGE(PIPELINE). It specifies a

URL of up to 255 characters to which the client's request should be redirected. This must be a complete URL, including scheme, host, and path components, and appropriate delimiters. CICS does not check that the URL is valid, so you must ensure that the destination exists and that the URL is specified correctly.

The description for the PATH attribute lists the characters that should be
excluded from a URL. These characters must not be used in the LOCATION
attribute. The exception is the # character, which can be used in the LOCATION
attribute as a separator before a fragment identifier which follows the URL.

The REDIRECTTYPE attribute is used to specify the type of redirection. If temporary or permanent redirection is specified, the URL in the LOCATION attribute is used for redirection. If no redirection is specified, the URL in the LOCATION attribute is ignored. You can use the SET URIMAP command to change the REDIRECTTYPE attribute and the LOCATION attribute.

MEDIATYPE(*type*)
This attribute is for USAGE(SERVER), where a static response is to be provided. It specifies the media type (data content) of the static response that CICS provides to the HTTP request, for example `image/jpg`, `text/html` or `text/xml`. Up to 56 characters can be used. The media type must contain exactly one forward slash (/). The media type can be entered in any case, but it is converted to lower case in the URIMAP definition.

The name for each formally recognized type of data content is defined by IANA. A list is available at http://www.iana.org/assignments/media-types/. CICS creates a Content-Type header for the response using the value of this attribute.

There is no default for this attribute, and it must be specified. If the MEDIATYPE attribute specifies a text type (such as a type that begins with `text/`, or a type that contains `+xml`), the CHARACTERSET and HOSTCODEPAGE attributes must also be specified so that code page conversion can take place. Text media types are identified by RFC 3023, which is available at http://www.ietf.org/rfc/rfc3023.txt.

For a dynamic (application-generated) response, this attribute is not used. The media type for the response is specified by the WEB SEND command.

PATH(*path*)
This attribute is for all USAGE options. It specifies the path component of the URI to which the URIMAP definition applies, which can be up to 255 characters. An example of a path is `software/htp/cics/index.html`. The minimum possible path is a / (forward slash), which represents the root of the URL structure for the specified host name.

The PATH attribute is specified in mixed case, and the case is preserved in the URIMAP definition. The PATH attribute must contain only the characters allowed in URIs. Specifically, the characters < > # % " { } | \ ^ [ ] ` and imbedded blanks should be excluded (except that % should be allowed when it introduces a valid hexadecimal escape sequence: that is, when it is followed by two valid hexadecimal digits in upper or lower case). The tilde character (~) cannot be specified in CICS and must be replaced by the corresponding hexadecimal escape sequence (%7E). CICS validates the use of characters at define time.

For URIMAP definitions relating to CICS as an HTTP server and Web services, if you want the URIMAP definition to match more than one path, you can use an asterisk as a wildcard character at the end of the path. For example, specifying the path `/software/htp/cics/*` would make the URIMAP definition match all requests whose path begins with the string to the left of the asterisk. Specifying a path of /* makes the URIMAP definition match any requests

directed to the host named in the HOST attribute. If an HTTP request is matched by more than one URIMAP definition, the most specific match is taken.

If a query component is present, and you want to apply the URIMAP definition to that specific query alone, you can include this as part of the path component. Include the question mark at the beginning of the string. The query string must contain only the characters allowed in URIs. A query string may not itself include an asterisk as a wildcard, but it may follow a path that includes an asterisk as a wildcard. If you do not include a query string in the URIMAP definition, any query string that is present in the HTTP request is automatically ignored for matching purposes.

For URIMAP definitions for CICS as an HTTP client, you cannot use an asterisk as a wildcard; you must specify the complete path for the request. If the URIMAP definition is referenced on a WEB OPEN command, this path becomes the default path for WEB SEND commands relating to that connection. If the URIMAP definition is referenced on a WEB SEND command, the path is used for that WEB SEND command, but note that the host attribute for that URIMAP definition must match the host specified on the WEB OPEN command for the connection.

PIPELINE(*name*)
> **This attribute is for USAGE(PIPELINE).** When a client makes an inbound Web service request to CICS with the URI specified by this URIMAP definition, PIPELINE specifies the 1-8 character name of the PIPELINE resource definition for the Web service. The PIPELINE resource definition provides information about the message handlers which act on the service request from the client.

> | **Acceptable characters:** |
> | --- |
> | A-Z 0-9 $ @ #

 Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase. |

PROGRAM(*name*)
> **This attribute is for USAGE(SERVER), where an application-generated response is to be provided.** It specifies the 1-8 character name of the user application program that composes the HTTP response. For CICS as an HTTP server, this attribute is required unless an analyzer or converter program is specified, or a template name or HFS file is specified, or redirection is specified.

> | **Acceptable characters:** |
> | --- |
> | A-Z 0-9 $ @ #

 Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase. |

REDIRECTTYPE({NONE|TEMPORARY|PERMANENT})
> **This attribute is for USAGE(SERVER) and USAGE(PIPELINE).** It specifies the type of redirection for requests that match this URIMAP definition. The URL specified by the LOCATION attribute is used for redirection when required.
> - NONE means that requests are not redirected. Any URL specified by the LOCATION attribute is ignored.
> - TEMPORARY means that requests are redirected on a temporary basis. The URL specified by the LOCATION attribute is used for redirection, and the status code used for the response is 302 (Found).

- PERMANENT means that requests are redirected permanently. The URL specified by the LOCATION attribute is used for redirection, and the status code used for the response is 301 (Moved Permanently).

You can use the SET URIMAP command to change the REDIRECTTYPE attribute and the LOCATION attribute.

SCHEME({<u>HTTP</u>|HTTPS})
This attribute is for all USAGE options. It specifies the scheme component of the URI to which the URIMAP definition applies, which is either HTTP (without SSL) or HTTPS (with SSL). Do not include the delimiters `://` (colon and two forward slashes) that follow the scheme component in the URI.

> **Note:** A URIMAP specifying the HTTP scheme accepts Web client requests made using either the HTTP scheme, or the HTTPS scheme. A URIMAP specifying the HTTPS scheme accepts only Web client requests made using the HTTPS scheme.

STATUS({<u>ENABLED</u>|DISABLED})
This attribute is for all USAGE options. It specifies whether the URIMAP definition is to be installed in an enabled or disabled state. The default is enabled.

TCPIPSERVICE(*name***)**
This attribute is for USAGE(SERVER) and USAGE(PIPELINE). It specifies the 1- to 8-character name of a TCPIPSERVICE resource definition, with PROTOCOL(HTTP), that defines an inbound port to which this URIMAP definition relates. If this attribute is not specified, the URIMAP definition applies to a request on any inbound ports.

> **Acceptable characters:**
> A-Z 0-9 $ @ #
>
> Unless you are using the CREATE command, any lowercase characters you enter are converted to uppercase.

When a URIMAP definition with HTTPS as the scheme matches a request that a Web client is making, CICS checks that the inbound port used by the request is using SSL. If SSL is not specified for the port, the request is rejected with a 403 (Forbidden) status code. When the URIMAP definition applies to all inbound ports, this check ensures that a Web client cannot use an unsecured port to access a secured resource. No check is carried out for a URIMAP definition that specifies HTTP as the scheme, so Web clients can use either unsecured or secured (SSL) ports to access these resources.

TEMPLATENAME(*name***)**
This attribute is for USAGE(SERVER), where a static response is to be provided. It specifies the 1-48 character name of a CICS document template that forms the body of the static response that is sent to the HTTP request from the Web client. It must be defined using a DOCTEMPLATE resource definition, and the TEMPLATENAME attribute of that definition specifies the name that is used in the URIMAP definition. See the *Application Programming Guide* for instructions on forming a CICS document template.

> **Acceptable characters:**
> A-Z a-z 0-9 $ @ # . / - _ % & ¢ ? ! : | " = ¬ , ; < >

Note: Resource level security is not applied to items delivered as a static response. If you need to apply access controls based on a user ID to an item delivered in this way, you need to deliver the material as an application-generated response instead.

If you want to use path matching, include an asterisk as a wildcard character at the end of the name of the CICS document template, and also at the end of the path specified by the PATH attribute. CICS takes the portion of each HTTP request's path that is covered by the wildcard character, and substitutes this as the last part of the template name.

For example, you could create a URIMAP definition with the PATH attribute specified as:

```
findout/about/*
```

and the TEMPLATENAME attribute specified as:

```
templates.facts.*
```

The URIMAP definition is used to process an incoming HTTP request

```
http://www.example.com/findout/about/fish.html
```

CICS appends `fish.html` to the template name specified in the URIMAP definition in place of the asterisk, so that the template

```
templates.facts.fish.html
```

is used to form the static response.

Note: Specifying an asterisk alone for the TEMPLATENAME attribute means that the selected template will have the same name as the part of the URL that corresponds to the wildcard character in the PATH attribute.

When the TEMPLATENAME attribute is specified, if a query string is present on the URI, but is not used in the PATH attribute, CICS automatically passes the content of the query string into the named CICS document template as a symbol list. If you want to use the content of the query string in the document template, you need to include appropriate variables in your document template to be substituted for the content of the query string.

TRANSACTION(*name*)
This attribute is for USAGE(SERVER), where an application-generated response is to be provided, and USAGE(PIPELINE). It specifies the 1-4 character name of an alias transaction that is to be used to run the user application that composes the HTTP response, or to start the pipeline.

Acceptable characters:
A-Z a-z 0-9 $ @ # . / - _ % & ¢ ? ! : | " = ¬ , ; < >

The default alias transaction is CWBA for USAGE(SERVER), or CPIH for USAGE(PIPELINE). You can select a different transaction name for the purposes of security, monitoring and accounting, or transaction class limitation. Whatever name you choose for the alias transaction, it must always run the same program, which is determined by the USAGE attribute. For USAGE(SERVER), the program is DFHWBA, which links to the application program named in the PROGRAM attribute of the URIMAP definition (or named by the analyzer program). For USAGE(PIPELINE), the program is DFHPIDSH, which starts the pipeline named in the PIPELINE attribute (and the Web service named in the WEBSERVICE attribute, if specified).

URIMAP(*name*)

> specifies the name of this URIMAP definition. The name can be up to eight characters in length. The attribute is specified in mixed case, and the case is preserved in the URIMAP definition.

> | **Acceptable characters:** |
> |---|
> | A-Z 0-9 $ @ # |

USAGE({**SERVER**|**CLIENT**|**PIPELINE**})

> Specifies whether this URIMAP definition is for CICS as an HTTP server (SERVER), CICS as an HTTP client (CLIENT), or a Web service (PIPELINE). The USAGE attribute governs which other attributes in the URIMAP definition can be used.

> Specifying SERVER creates a URIMAP definition for CICS as an HTTP server. This type of URIMAP definition is used to map the URI of an incoming HTTP request from a Web client, to CICS resources. An application-generated response or a static response can be provided.

> Specifying CLIENT creates a URIMAP definition for CICS as an HTTP client. This type of URIMAP definition is used when CICS makes a request for an HTTP resource on a server, so that you can avoid identifying the URI in your application program.

> Specifying PIPELINE creates a URIMAP definition for a Web service. This type of URIMAP definition is used for an inbound Web service request (that is, a request by which a client invokes a Web service in CICS). The URI of the incoming request is associated with WEBSERVICE and PIPELINE resources, which specify the processing that is to be performed on the message.

USERID(*id*)

> **This attribute is for USAGE(SERVER), where an application-generated response is to be provided, and USAGE(PIPELINE).**

> USERID specifies a 1- to 8-character default user ID that can be used by any client. For an application-generated response or a web service, the alias transaction is attached under this user ID.

> When authentication is required for the connection, so that CICS requests an authenticated user ID directly from the client, the default user ID that you specify in the URIMAP definition is not used. The authenticated user ID of the client is used instead, or if authentication fails, the request is rejected. Authentication procedures are specified by the AUTHENTICATE attribute of the TCPIPSERVICE definition for the connection.

> If ANALYZER(YES) is specified, a user ID from a URIMAP definition or from the client can be changed by the analyzer program, or the analyzer program can set a user ID. If no user ID is specified by any of these means, the default user ID is the CICS default user.

WEBSERVICE(*name*)

> **This attribute is for USAGE(PIPELINE).** When a client makes an inbound Web service request to CICS with the URI specified by this URIMAP definition, WEBSERVICE specifies the name of the Web service. This can be the 1-8 character name of a WEBSERVICE resource definition, or a name up to 32 characters (in mixed case) representing a Web service generated by the CICS Web services assistant.

Acceptable characters:
`A-Z a-z 0-9 $ @ # . / - _ % & ¢ ? ! :	" = ¬ , ; < >`

A Web service definition defines aspects of the run time environment for a CICS application program deployed in a Web services setting, where the mapping between application data structure and SOAP messages has been generated using CICS-supplied tools.

Changes to the application programming interface (General CICS Web support enhancements)

New and changed commands

There are new **EXEC CICS WEB** commands for CICS as an HTTP client. Also, some commands have been modified for CICS as an HTTP client, but are unchanged for CICS as an HTTP server. The new and changed client commands are described in "Changes to the application programming interface (HTTP client requests)" on page 86.

The **EXEC CICS WEB SEND** and **EXEC CICS WEB RECEIVE** commands are enhanced for HTTP/1.1 support when used by CICS as an HTTP server, and are described in "Changes to the application programming interface (HTTP/1.1 support)" on page 119.

The following new **EXEC CICS** commands can be used for both CICS as an HTTP server, and CICS as an HTTP client:

- **EXEC CICS WEB PARSE URL**
- **EXEC CICS CONVERTTIME**

The following **EXEC CICS** commands have been modified for both CICS as an HTTP server, and CICS as an HTTP client:

- **EXEC CICS WEB EXTRACT**
- **EXEC CICS FORMATTIME**

WEB PARSE URL

Breaks down a URL string into its component parts.

```
WEB PARSE URL

►►──WEB──PARSE URL(data-value)──URLLENGTH(data-value)─────────────────────►

 ►─┬──────────────────────────┬──┬──────────────────────────────────────┬──►
   └─SCHEMENAME(data-area)─────┘  └─HOST(data-area)──HOSTLENGTH(data-area)─┘

 ►─┬──────────────────────┬──┬────────────────────────────────────────┬──►
   └─PORTNUMBER(data-area)─┘  └─PATH(data-area)──PATHLENGTH(data-area)──┘

 ►─┬──────────────────────────────────────────────────┬──►◄
   └─QUERYSTRING(data-area)──QUERYSTRLEN(data-area)─────┘
```

Conditions: INVREQ, LENGERR

This command is threadsafe.
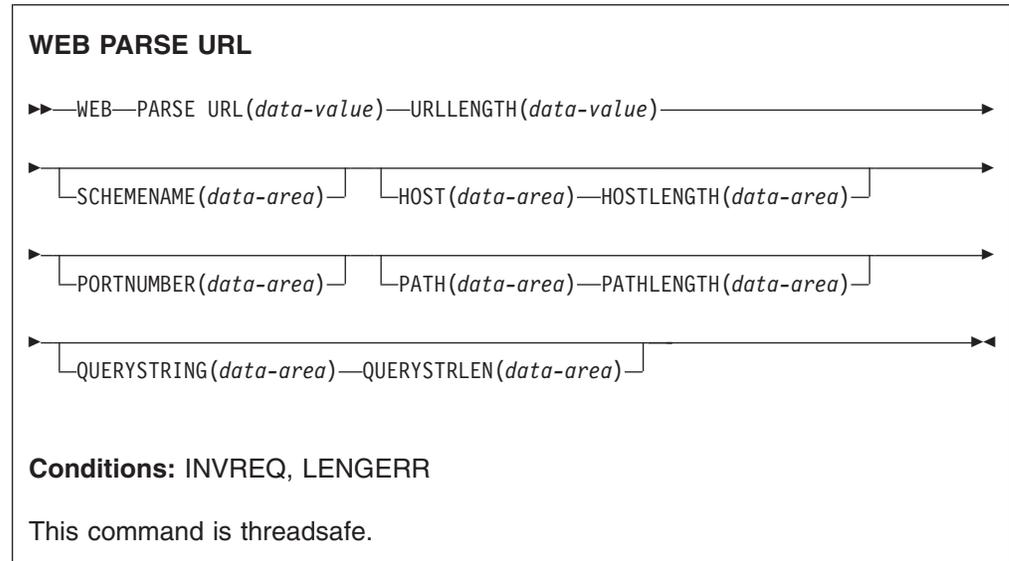
Description

WEB PARSE URL enables you to break down a URL string into its component parts: scheme, host, port, path and query string. You can use this process to examine the construction of the URL, and to separate out the components. The returned information can be used in the WEB OPEN command to open a client connection to the host named in the URL.

Any escape sequences found in the URL are checked for validity. An escape sequence consists of the percent character (%) followed by two hexadecimal characters. Valid hexadecimal characters are the digits 0 to 9 and the letters A to F.

Note that where the string input to the WEB PARSE URL command has been delimited in the correct way for a URL, the command does not detect invalid content, such as a host name that does not represent an existing host on the Internet, or a character that is not permitted in a URL.

Options

HOST*(data-area)*
    returns the host component of the URL. This can be either an alphanumeric host name or a numeric IP address. If a port number was specified explicitly in the URL, this is returned separately as the PORTNUMBER option.

    An IPv4 address can be used as a host name in the WEB OPEN command, but IPv6 addresses are **not** supported. IPv6 addresses are rejected as invalid by the WEB PARSE URL command because they do not conform to the expected structure.

HOSTLENGTH*(data-area)*
    specifies the length of the buffer supplied on the HOST option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the host name). 116 characters is suggested as an appropriate size

to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

PATH(*data-area*)
> returns the path component of the URL.

PATHLENGTH(*data-area*)
> specifies the length of the buffer supplied on the PATH option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the path component of the URL). 256 characters is suggested as an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

PORTNUMBER(*data-area*)
> returns (as a fullword binary data area) the port number that is specified in, or appropriate for, the URL. Port numbers are sometimes specified explicitly in a URL, following the host name. However, well-known port numbers for a service are normally omitted from a URL. If the port number is not present in the URL, the WEB PARSE URL command identifies and returns it based on the scheme. For HTTP, the well-known port number is 80, and for HTTPS, the well-known port number is 443. If a port number is returned which is not the default for the scheme, you need to specify the port number explicitly to gain access to the URL (for example, if you are using this information in a WEB OPEN command).

QUERYSTRING(*data-area*)
> returns the query string from the URL. The query string is the value or values encoded after the question mark (?) delimiting the end of the path. The query string is returned in its escaped form.

QUERYSTRLEN(*data-area*)
> specifies the length of the buffer supplied on the QUERYSTRING option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the query string). 256 characters is suggested as an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

SCHEMENAME(*data-area*)
> returns the scheme component of the URL, as a 16-character data area. Only the HTTP and HTTPS schemes (the HTTP protocol with and without SSL) are supported by CICS and can be used in a WEB OPEN command.

> The scheme name is always returned in upper case.

URL(*data-value*)
> specifies the complete URL string.

URLLENGTH(*data-value*)
> specifies the length of the buffer containing the URL string, as a fullword binary value.

WEB EXTRACT

Obtain information about an HTTP request that has been made to CICS as an HTTP server, or about a connection between an Internet server and CICS as an HTTP client.

WEB EXTRACT (CICS as an HTTP server)

```
►►──WEB──EXTRACT────────────────────────────────────────────────►
              └─SCHEME(cvda)─┘

►─────────────────────────────────────────────────────────────────►
   └─HOST(data-area)──HOSTLENGTH(data-value)─┘

►─────────────────────────────────────────────────────────────────►
   └─HTTPMETHOD(data-area)──METHODLENGTH(data-area)─┘

►─────────────────────────────────────────────────────────────────►
   └─HTTPVERSION(data-area)──VERSIONLEN(data-area)─┘

►─────────────────────────────────────────────────────────────────►
   └─PATH(data-area)──PATHLENGTH(data-area)─┘  └─PORTNUMBER(data-area)─┘

►─────────────────────────────────────────────────────────────────◄
   └─QUERYSTRING(data-area)──QUERYSTRLEN(data-area)─┘  └─REQUESTTYPE(cvda)─┘
```
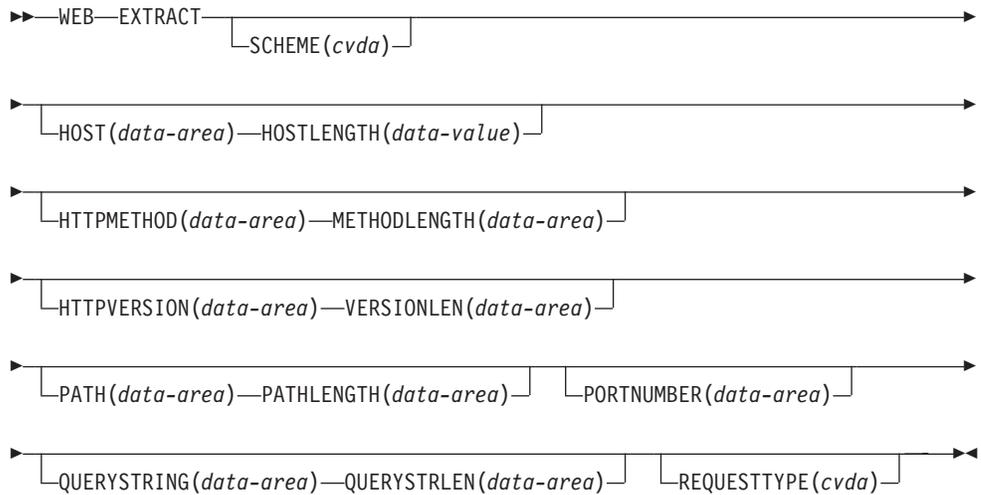
Conditions: ILLOGIC, INVREQ, LENGERR, NOTOPEN

This command is threadsafe.

WEB EXTRACT (CICS as an HTTP client)

```
►►──WEB──EXTRACT──SESSTOKEN(data-area)───────────────────────────►
                                     └─SCHEME(cvda)─┘

►─────────────────────────────────────────────────────────────────►
   └─HOST(data-area)──HOSTLENGTH(data-value)─┘

►─────────────────────────────────────────────────────────────────►
   └─HTTPVERSION(data-area)──VERSIONLEN(data-area)─┘

►─────────────────────────────────────────────────────────────────►
   └─PATH(data-area)──PATHLENGTH(data-area)─┘  └─PORTNUMBER(data-area)─┘

►─────────────────────────────────────────────────────────────────◄
   └─URIMAP(data-area)─┘
```
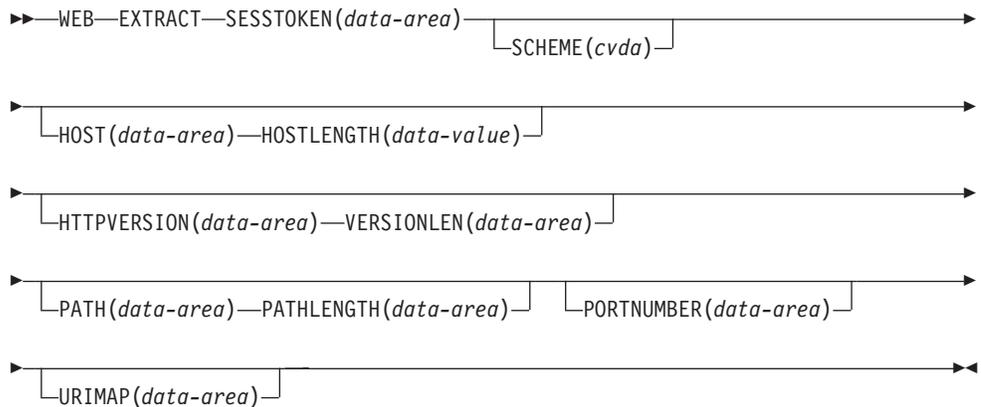
Conditions: ILLOGIC, INVREQ, LENGERR, NOTOPEN

This command is threadsafe.

Options

HOST(*data-area*)
>   For CICS as an HTTP server, this option specifies a buffer to contain the host
>   component of the URL, as specified either in the Host header field for the

request, or in the request line (if an absolute URI was used for the request). The port number is presented separately using the PORTNUMBER option.

For CICS as an HTTP client (with the SESSTOKEN option), this option specifies a buffer to contain the host name of the server in the connection identified by the SESSTOKEN option. The port number is presented separately using the PORTNUMBER option.

HOSTLENGTH*(data-area)*
specifies the length of the buffer supplied on the HOST option, as a fullword binary variable, and is set to the actual length of the data returned to the application. 116 characters is an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

HTTPMETHOD*(data-area)*
For CICS as an HTTP server, this option specifies a buffer to contain the HTTP method string on the request line of the message.

This option is not relevant for CICS as an HTTP client.

HTTPVERSION*(data-area)*
For CICS as an HTTP server, this option specifies a buffer to contain the HTTP version for the Web client, as stated on its request.

For CICS as an HTTP client (with the SESSTOKEN option), this option specifies a buffer to contain the HTTP version of the server in the connection identifed by the SESSTOKEN option.

"1.1" indicates HTTP/1.1, and "1.0" indicates HTTP/1.0.

Note: CICS does not make any special provision for a server or Web client that is below HTTP/1.0 level. CICS behaves as though they were at HTTP/1.0 level, and returns "1.0" as the HTTP version.

If your application program writes HTTP headers that might be unsuitable for a Web client or server at HTTP/1.0 level, or if you intend to send chunked information to the Web client or server (which cannot be received by a client or server at HTTP/1.0 level), your application program should check the HTTP version information.

METHODLENGTH*(data-area)*
specifies the length of the buffer supplied on the HTTPMETHOD option, as a fullword binary variable, and is set to the actual length of the data returned to the application. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

PATH*(data-area)*
For CICS as an HTTP server, this option specifies a buffer to contain the path specified in the request line of the message.

For CICS as an HTTP client (with the SESSTOKEN option), this option specifies a buffer to contain the default path that applies to requests made using the connection. If a URIMAP definition was specified on the WEB OPEN command for the connection, the default path is the path specified in the URIMAP definition. Otherwise, the default path is a single forward slash.

PATHLENGTH*(data-area)*
specifies the length of the buffer supplied on the PATH option, as a fullword binary variable, and is set to the actual length of the data returned to the

application. 256 characters is an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

PORTNUMBER(*data-area*)
For CICS as an HTTP server, this option returns a data area containing the port number specified in the request line of the message.

For CICS as an HTTP client (with the SESSTOKEN option), this option returns a data containing the port number used to access the server in the connection specified by the SESSTOKEN option.

The value returned in the data area is a fullword binary value.

Well-known port numbers for a service are normally omitted from the URL. If the port number is not present in the URL, the WEB EXTRACT command identifies and returns it based on the scheme. For HTTP, the well-known port number is 80, and for HTTPS, the well-known port number is 443. If a port number is returned which is not the default for the scheme, you need to specify the port number explicitly to gain access to the URL (for example, if you are using this information in a WEB OPEN command).

QUERYSTRING(*data-area*)
For CICS as an HTTP server, this option specifies a buffer to contain the query string on the request line of the message. The query string is the value or values encoded after the question mark (?) delimiting the end of the path. The query string is returned in its escaped form.

This option is not relevant for CICS as an HTTP client.

QUERYSTRLEN(*data-area*)
specifies the length of the buffer supplied on the QUERY option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the query string). 256 characters is an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

REQUESTTYPE(*cvda*)
For CICS as an HTTP server, this option specifies the type of request received. This option is not relevant for CICS as an HTTP client. CVDA values are:

HTTPYES
indicates an HTTP request.

HTTPNO
indicates a non-HTTP request.

SCHEME(*cvda*)
For both CICS as an HTTP server, and CICS as an HTTP client (with the SESSTOKEN option), this option returns the scheme used for the connection between CICS and the Web client or server. CVDA values are:

HTTP   is the HTTP protocol, without SSL.

HTTPS
is the HTTPS protocol, which is HTTP with SSL.

SESSTOKEN(*data-value*)
For CICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between CICS and a server. This value is returned by a WEB OPEN command for CICS as an HTTP client. "Session tokens" in the *CICS Internet Guide* explains the

use of the session token. For the WEB EXTRACT command, information is returned about the specified connection.

This option is not relevant for CICS as an HTTP server.

URIMAP(*data-value*)
For CICS as an HTTP client (with the SESSTOKEN option), this option returns the 8-character name (in mixed case) of any URIMAP definition that was specified on the WEB OPEN command to open the connection specified by the SESSTOKEN option. The INQUIRE URIMAP command can be used to find information about the attributes of this URIMAP definition.

This option is not relevant for CICS as an HTTP server.

VERSIONLEN(*data-area*)
specifies the length of the buffer supplied on the HTTPVERSION option, as a fullword binary variable, and is set to the actual length of the data returned to the application.

CONVERTTIME
Converts an architected date and time stamp string to the ABSTIME format.

CONVERTTIME

►►──CONVERTTIME──DATESTRING(*data-area*)──ABSTIME(*data-area*)──────────────►◄

Conditions: INVREQ, LENGERR

This command is threadsafe.

Description

CONVERTTIME analyzes three different date and time stamp formats which are commonly used on the Internet, and converts them to the ABSTIME (absolute date and time) format.

ABSTIME format gives the time, in packed decimal, since 00:00 on 1 January 1900 (in milliseconds rounded to the nearest hundredth of a second). The FORMATTIME command can be used to change this into other formats.

The architected date and time stamp string formats recognized by the CONVERTTIME command are:

RFC 1123 format
The preferred standard format for date and time stamps for the HTTP protocol, as specified in RFC 1123. An example of a date and time stamp in this format is "Tue, 01 Apr 2003 10:01:02 GMT".

RFC 850 format
An older date and time stamp format for the Internet. An example of a date and time stamp in this format is "Tuesday, 01-Apr-03 10:01:02 GMT".

Important: Because the year has only two digits in this format, CICS uses the assumption that the years are in the range 1970 to 2069. In the example above, CICS would assume that the date of the

document was 1 April 2003. Given the date and time stamp "Thursday, 13-Feb-98 15:30:00 GMT", CICS would assume that the date of the document was 13 February 1998. Be aware of this when coding your application, if you think that you could receive date and time stamps in this format.

ASCtime format

A date and time stamp format output from the C ASCtime function. An example of a date and time stamp in this format is "Tue Apr 1 10:01:02 2003".

Options

DATESTRING(*data-area*)

specifies a 64-character data-area to contain the architected date and time stamp string. You can supply a string in any of the formats recognized by the
command, and you do not need to specify which format is used.If the date and
time stamp string is in the RFC 1123 format, which is always at GMT, the date
and time are converted to local time for the ABSTIME which is returned.

ABSTIME(*data-area*)

specifies a data-area to receive the converted date and time stamp in ABSTIME format. For the format of this data-area, see the description of the ASKTIME command. If the date and time stamp was not in a recognized format, no ABSTIME is returned.

Changes to options on `EXEC CICS WEB` commands

The DATESTRING and STRINGFORMAT options are added to the FORMATTIME command to return a date and time stamp in a format for use on the Internet.

FORMATTIME

The DATESTRING and STRINGFORMAT options are added to the `EXEC CICS FORMATTIME` command. These options enable you to convert a date and time stamp in ABSTIME format to an architected date and time stamp string. The only format provided by the STRINGFORMAT option at present is the format that complies with the RFC 1123 standard, which is suitable for use on the Internet.

DATESTRING(*data-area*)

specifies the 64-character user field where CICS is to return the architected date and time stamp string in the format specified by the STRINGFORMAT option. If STRINGFORMAT is not specified, the default format provided is the RFC 1123 format (RFC1123).

STRINGFORMAT(*cvda*)

specifies the format for the architected date and time stamp string returned in DATESTRING. The only CVDA value available at present is:

RFC1123

specifies the RFC 1123 format, which is suitable for use on the Internet. This date and time stamp string contains the day, date, and 24-hour clock time at GMT, for example "Tue, 01 Apr 2003 10:01:02 GMT".

Changes to the system programming interface

The HFSFILE option is added to the `INQUIRE DOCTEMPLATE` and `CREATE DOCTEMPLATE` commands:

HFSFILE(*filename*)
>
> specifies the fully qualified (absolute) or relative name of the HFS file of the z/OS UNIX System Services HFS file where the template resides. This can be up to 255 characters in length.

The following commands are provided to manage URIMAP definitions:

- **CREATE URIMAP**
- **DISCARD URIMAP**
- **INQUIRE URIMAP**
- **SET URIMAP**

CREATE URIMAP command

Use the **CREATE URIMAP** command to dynamically create a URIMAP in your CICS region. The attributes you can specify on this command are described in: "URIMAP definition attributes" on page 136.

DISCARD URIMAP command

Use the **DISCARD URIMAP** to remove a URIMAP from your CICS region. The URIMAP must be disabled before it can be discarded.

INQUIRE URIMAP

Retrieve information about URIMAP resources in the local system.

```
INQUIRE URIMAP

►►──INQUIRE URIMAP(data-area)──┬─────────────────────────────┬──►◄
                               ├─ENABLESTATUS(cvda)──────────┤
                               ├─USAGE(cvda)─────────────────┤
                               ├─HOST(data-area)─────────────┤
                               ├─PATH(data-area)─────────────┤
                               ├─MEDIATYPE(data-area)────────┤
                               ├─CHARACTERSET(data-area)─────┤
                               ├─HOSTCODEPAGE(data-area)─────┤
                               ├─TEMPLATENAME(data-area)─────┤
                               ├─HFSFILE(data-area)──────────┤
                               ├─TCPIPSERVICE(data-area)─────┤
                               ├─ANALYZERSTAT(cvda)──────────┤
                               ├─CONVERTER(data-area)────────┤
                               ├─TRANSACTION(data-area)──────┤
                               ├─PROGRAM(data-area)──────────┤
                               ├─PIPELINE(data-area)─────────┤
                               ├─WEBSERVICE(data-area)───────┤
                               ├─USERID(data-area)───────────┤
                               ├─CERTIFICATE(data-area)──────┤
                               ├─CIPHERS(data-area)──────────┤
                               ├─NUMCIPHERS(data-area)───────┤
                               ├─REDIRECTTYPE(cvda)──────────┤
                               └─LOCATION(data-area)─────────┘

Conditions: END, ILLOGIC, NOTAUTH, NOTFND

This command is threadsafe.
```

Description

The INQUIRE URIMAP command allows you to retrieve information about a particular URIMAP definition. The USAGE attribute of a URIMAP definition determines which other attributes are specified in that URIMAP definition, and sometimes determines the meaning of a particular attribute.

Browsing

You can also browse through all the URIMAP definitions installed in the region, using the browse options (START, NEXT, and END) on INQUIRE URIMAP commands.

Options

URIMAP(*data-value*)
    specifies the 8-character name of a URIMAP definition.

ANALYZERSTAT*(cvda)*
> returns a CVDA value indicating whether the analyzer program associated with the TCPIPSERVICE definition is to be run. CVDA values are:

> **ANALYZER**
>> The analyzer program is to be run.

> **NOANALYZER**
>> The analyzer program is not to be run.

> This attribute is for USAGE(SERVER). For all other usage types it is forced to NO.

CERTIFICATE*(data-area)*
> returns a 32-character data area containing the label of the certificate that is to be used as the SSL client certificate for the HTTP request by CICS as an HTTP client. This attribute is for USAGE(CLIENT).

CHARACTERSET*(data-area)*
> returns a 40-character data area containing the name of the character set to be used for the static response. This attribute is for USAGE(SERVER).

CIPHERS*(data-area)*
> returns a 56-character data area containing the list of cipher suites specified for the URIMAP definition. The list of cipher suites is used to negotiate SSL connections. This attribute is for USAGE(CLIENT).

CONVERTER*(data-area)*
> returns the 8-character name of a converter program that performs conversion or other processing for CICS as an HTTP server. This attribute is for USAGE(SERVER).

ENABLESTATUS*(cvda)*
> returns a CVDA value indicating the status of this URIMAP definition. CVDA values are:

> **ENABLED**
>> The URIMAP definition is enabled.

> **DISABLED**
>> The URIMAP definition is disabled. A URIMAP definition with this status can be discarded.

> **DISABLEDHOST**
>> The URIMAP definition is unavailable because the virtual host of which it is a part has been disabled. The SET HOST command can be used to re-enable all the URIMAP definitions that make up the virtual host. A URIMAP definition with this status cannot be discarded.

HFSFILE*(data-area)*
> returns a 255-character data area containing fully qualified (absolute) or relative name of a z/OS UNIX System Services HFS file that forms a static response. This attribute is for USAGE(SERVER).

HOST*(data-area)*
> returns a 116-character data area containing the host component of the URI to which the URIMAP definition applies (for example, `www.example.com`). This attribute is for any usage type.

HOSTCODEPAGE*(data-area)*
> returns a 10-character data area containing the 1-10 character name of the IBM

code page (EBCDIC) in which the text document that forms the static response is encoded. This attribute is for USAGE(SERVER).

LOCATION*(data-area)*
returns a 255-character area containing a URL to which matching HTTP requests from Web clients are redirected. Redirection is activated by the setting specified by the REDIRECTTYPE option. This attribute is for USAGE(SERVER) or USAGE(PIPELINE).

MEDIATYPE*(data-area)*
returns a 56-character data area containing a description of the data content of the static response. This attribute is for USAGE(SERVER).

NUMCIPHERS*(data-area)*
returns a halfword binary value containing the number of cipher codes in the CIPHERS list. The ciphers are used to negotiate encryption levels as part of the SSL handshake. This attribute is for USAGE(CLIENT).

PATH*(data-area)*
returns a 255-character data area containing the path component of the URL to which the URIMAP definition applies (for example, software/htp/cics/index.html). This attribute is for any usage type.

PIPELINE*(data-area)*
returns the 8-character name of the PIPELINE resource definition for the Web service. The PIPELINE resource definition provides information about the message handlers which act on the service request from the client. This attribute is for USAGE(PIPELINE).

PROGRAM*(data-area)*
returns the 8-character name of the application program that composes an application-generated response to the HTTP request. This attribute is for USAGE(SERVER).

SCHEME*(cvda)*
returns a CVDA value indicating the scheme component of the URI. CVDA values are:

HTTP HTTP without SSL.

HTTPS
HTTP with SSL.

This attribute is for any usage type.

TCPIPSERVICE*(data-area)*
returns the 1- to 8-character name of the TCPIPSERVICE definition that specifies an inbound port to which this URIMAP definition relates. If this is not specified, the URIMAP definition applies to a request on any inbound ports. This attribute is for USAGE(SERVER) or USAGE(PIPELINE).

TEMPLATENAME*(data-area)*
returns a 48-character data area containing the name of a CICS document template that is used to form a static response. This attribute is for USAGE(SERVER).

TRANSACTION*(data-area)*
returns the 4-character name of an alias transaction to run the user application that composes a response to the HTTP request. This attribute is for USAGE(SERVER) or USAGE(PIPELINE).

REDIRECTTYPE(*cvda*)
> returns a CVDA value indicating the type of redirection for requests that match this URIMAP definition. The URL for redirection is specified by the LOCATION option. This attribute is for USAGE(SERVER) or USAGE(PIPELINE). CVDA values are:

> **NONE** Requests are not redirected. Any URL specified by the LOCATION option is ignored.

> **TEMPORARY**
>> Requests are redirected on a temporary basis. The status code used for the response is 302 (Found).

> **PERMANENT**
>> Requests are redirected permanently. The status code used for the response is 301 (Moved Permanently).

USAGE(*cvda*)
> returns a CVDA value indicating the purpose of this URIMAP definition. CVDA values are:

> **SERVER**
>> A URIMAP definition for CICS as an HTTP server. This type of URIMAP definition is used to map the URL of an incoming HTTP request from a Web client, to CICS resources. An application-generated response or a static response can be provided.

> **CLIENT**
>> A URIMAP definition for CICS as an HTTP client. This type of URIMAP definition is used when CICS makes a client request for an HTTP resource on a server.

> **PIPELINE**
>> A URIMAP definition for a Web service. This type of URIMAP definition is used to specify the processing that is to be performed on a request by which a client invokes a Web service in CICS.

USERID(*data-area*)
> returns the 8-character user ID under which the alias transaction is attached. This attribute is for USAGE(SERVER) or USAGE(PIPELINE).

WEBSERVICE(*data-area*)
> returns the name of a Web service. This can be the 1-8 character name of a WEBSERVICE resource definition, or a name up to 32 characters representing a Web service generated by the CICS Web services assistant. This defines aspects of the run time environment for a CICS application program deployed in a Web services setting. This attribute is for USAGE(PIPELINE).

SET URIMAP
Enables or disables a URIMAP definition, and applies or removes redirection for a URIMAP definition.

```
SET URIMAP

►►──SET URIMAP(data-area)──────────────────────────────────────────►
                        └─ENABLESTATUS(cvda)─┘

►──────────────────────────────────────────────────────────────►◄
  └─REDIRECTTYPE(cvda)──LOCATION(data-area)─┘
```
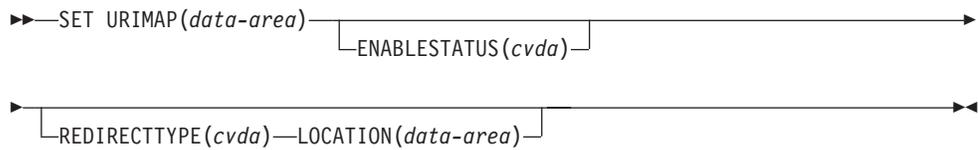
Conditions: INVREQ, NOTAUTH, NOTFND

This command is threadsafe.

Description

The SET URIMAP command allows you to:
- Enable or disable a URIMAP definition.
- Set redirection for matching HTTP requests, and specify a URL to which the requests are redirected. You can use this command to apply redirection to an existing URIMAP definition, for example if the application that would normally respond to the HTTP request is unavailable. You can also use this command to remove redirection from a URIMAP definition.

Options

ENABLESTATUS(*cvda*)

Sets the URIMAP definition to enabled or disabled status. CVDA values are:

ENABLED

The URIMAP definition can be accessed by applications.

DISABLED

The URIMAP definition cannot be accessed by applications. A URIMAP definition has to be disabled before it can be reinstalled or discarded.

LOCATION(*data-area*)

Specifies a URL of up to 255 characters, to which matching HTTP requests from Web clients can be redirected. This must be a complete URL, including scheme, host, and path components, and appropriate delimiters. CICS does not check that the URL is valid, so you must ensure that the destination exists and that the URL is specified correctly.

The REDIRECTTYPE option is used to specify the type of redirection. If temporary or permanent redirection is specified, the URL in the LOCATION attribute is used for redirection. If NONE is specified, the URL in the LOCATION option is ignored.

REDIRECTTYPE(*cvda*)

Specifies the type of redirection for requests that match this URIMAP definition. The URL for redirection is specified by the LOCATION option. CVDA values are:

NONE Requests are not redirected. Any URL specified by the LOCATION option is ignored.

TEMPORARY

Requests are redirected on a temporary basis. The HTTP status code used for the response is 302 (Found).

PERMANENT

Requests are redirected permanently. The HTTP status code used for the response is 301 (Moved Permanently).

Changes to CEMT

The following new commands are added to the CEMT transaction:

*   **INQUIRE URIMAP**
*   **SET URIMAP**
*   **DISCARD URIMAP**
*   **INQUIRE HOST**
*   **SET HOST**

There are changes to the following commands:

*   **INQUIRE TCPIPSERVICE**
*   **SET TCPIPSERVICE**
*   **INQUIRE DOCTEMPLATE**

CICS Supplied Transactions has information about these new and changed commands.

Changes to CICS-supplied transactions

New CICS-supplied transaction CWXU

In CICS Transaction Server for z/OS, Version 3 Release 1, processing for HTTP requests and processing for non-HTTP requests are kept separate. This ensures that CICS can perform basic acceptance checks on HTTP requests and responses, as described in Chapter 4, "CICS Web support upgrade to HTTP/1.1," on page 111, and that non-HTTP requests are not subjected to these checks. Processing for non-HTTP requests must now be carried out under the user-defined (USER) protocol, which is specified on the TCPIPSERVICE definition for the port that receives the requests.

The new CICS-supplied transaction CWXU, the CICS Web user-defined protocol attach transaction, is the default when the protocol is defined as USER. CWXU executes the CICS program DFHWBXN. The DFHCURDI sample includes a sample definition for CWXU. An alternative transaction that executes DFHWBXN may be used, with the exception of the other default transactions that are defined for protocols on the TCPIPSERVICE resource definition.

CWXU is a RACF Category 1 transaction.

Changes to CWXN

There are several changes to the processing carried out by the CICS-supplied transaction CWXN, the Web attach transaction. The most significant of these are:

*   If a matching URIMAP definition is found for an HTTP request, CWXN now invokes the analyzer program only if instructed to do so by the URIMAP definition.
*   Where the HTTP version of the request is HTTP/1.1, CWXN carries out some of the responsibilities of an HTTP server by performing some basic acceptance

checks on the request. In response to these checks, CWXN might take action to return a response to the request without involving a user-written application program.

- CWXN pre-processes chunked and pipelined messages received from a Web client, so that user-written applications do not have to perform this processing.
  - Chunked messages are single messages split up and sent as a series of smaller messages (chunks). CWXN receives and assembles the chunks of the message to create a single HTTP request. CWXN checks that the message is complete before passing it to the user application. The user application can then process the request like any other HTTP request.
  - Pipelined messages are multiple messages sent in sequence, where the sender does not wait for a response after each message sent. A server must respond to these messages in the order that they are received. To ensure this, CWXN holds pipelined requests and releases them one at a time to the user application. The user application must send a response to the first request before receiving the next request from CWXN.
- Persistent connections are now the default behavior. The connection is only closed if the Web client requests closure, or if the timeout period is reached, or if the Web client is an HTTP/1.0 client that does not send a Keep-Alive header.
- Before CICS Transaction Server for z/OS, Version 3 Release 1, if a Web client and CICS had a persistent connection, the CWXN transaction would remain in the system for the duration of the persistent connection. Now, the CWXN transaction terminates after each request from the Web client has been passed to the alias transaction (CWBA or another transaction), or after the static response has been delivered. The Sockets listener task monitors the socket and initiates a new instance of CWXN for each request on the persistent connection. This behavior, known as an asynchronous receive, avoids the possibility of a deadlock in a situation where the maximum task specification (MXT) has been reached. It also means that the maximum number of concurrent connections between CICS and Web clients is no longer limited by the MXT value, but can in theory be up to 64000. In terms of system activity, if you used persistent connections before CICS Transaction Server for z/OS, Version 3 Release 1, you should now see an increased transaction rate, but a decrease in the number of concurrent tasks.

Priorities for CICS Web support transactions (CWXN, CWXU, CWBA or other alias transactions)

If you set the priority of the CWXN or CWXU transaction higher than the priority of the alias transactions used for application-generated responses (such as CWBA), this can result in a build-up of requests that have been received but not yet processed, which may lead to a short-on-storage situation.

The default priorities for the CWXN or CWXU transaction are set to 1, and the default priority for the CICS-supplied CWBA transaction is also set to 1. You can adjust these priorities depending on your workload. Make sure the priorities of the alias transactions used for application-generated responses (like CWBA) are equal to, or higher than, the priority of the transactions associated with Web attach tasks (like CWXN or CWXU). Bear this in mind if you are setting up your own transaction definitions in place of the CICS-supplied defaults.

Changes to user-replaceable programs
Changes to analyzer programs

Use of an analyzer program for CICS Web support processing for individual HTTP requests is now optional. URIMAP definitions can be used to match the URLs of requests to the application program that processes them, and they can specify the use of a converter program and an alias transaction. If these are the only tasks performed by an analyzer program in your existing CICS Web support architecture, you can replace its function in request processing paths with a URIMAP definition.

You might have an existing analyzer program from an earlier CICS release that provides additional functions which you require during request processing, such as passing data to a converter program, or modifying code page conversion for non-Web-aware application programs that use a converter program. If this is the case, you can continue to use an analyzer program instead of a URIMAP definition for the relevant requests, or you can combine it with a URIMAP definition (by specifying the ANALYZER(YES) option).

When an analyzer program is used with a URIMAP definition, elements of the URIMAP definition (such as the name of the application program that handles the request) are passed to the analyzer program as input. An existing or new analyzer program can be used to make dynamic changes to these elements. You can also use an analyzer program to introduce monitoring or audit actions into the process.

An analyzer program must still be specified for each TCPIPSERVICE resource definition that is used for CICS Web support. The analyzer program specified for a TCPIPSERVICE definition is invoked to handle an HTTP request if CICS does not find a matching URIMAP definition for the request. This could be caused by a user error in typing a request URL, or because the appropriate URIMAP definition is not installed. (If the URIMAP definition exists but is disabled, the request is handled by the Web error program, not the analyzer program.)

A new CICS-supplied default analyzer program is provided to supply this error handling function for TCPIPSERVICE resource definitions that are used for CICS Web support. DFHWBAAX takes no action if a matching URIMAP definition is found. If no match is found, it gives control to the new user-replaceable Web error program DFHWBERX to produce an error response. DFHWBAAX is suitable for use where all of the requests using the port are handled using URIMAP definitions. It does not provide support for requests using the URL format that CICS Web support used before CICS TS 3.1.

The CICS-supplied sample analyzer program DFHWBADX is still provided. DFHWBADX, or your own customized version of it, is suitable for use if you need to provide basic support for both requests using URIMAP definitions, and requests using the URL format that CICS Web support used before CICS TS 3.1.

As supplied, DFHWBADX does not perform any analysis of a request when a matching URIMAP definition has been found for the request, even if the URIMAP specifies ANALYZER(YES). This means that the settings specified in the URIMAP definition for the alias transaction, converter program and application program are automatically accepted and used to determine subsequent processing stages. DFHWBADX uses the `wbra_urimap` input parameter to test for the presence of a URIMAP definition.

There are several changes to the input and output parameters specified in an analyzer program's COMMAREA:

- A new input parameter `wbra_urimap` is provided to identify when a matching URIMAP definition is involved in the processing path for the request.

- New input parameters `wbra_hostname_ptr`, `wbra_hostname_length`, `wbra_querystring_ptr`, and `wbra_querystring_length` provide the host name and query string specified on the Web client's request. An analyzer program can examine this information in addition to the path component of the URL to decide on subsequent processing stages, and to distinguish between virtual hosts (multiple hosts at the same IP address).

- The parameters `wbra_alias_tranid`, `wbra_converter_program`, `wbra_server_program` and `wbra_userid` are now input parameters as well as output parameters. When a URIMAP definition is used, the TRANSACTION, CONVERTER, PROGRAM, and USERID attributes (respectively) of the URIMAP definition are passed to the analyzer program as these input parameters, and the analyzer program can choose to override these.

- Two new output parameters are provided for code page conversion. The character set used by the Web client can be specified by the `wbra_characterset` parameter, and the `wbra_hostcodepage` parameter specifies the host code page suitable for the application program. CICS uses these parameters to carry out code page conversion before passing the request to the converter program (if used) or to the application. The output parameters are functionally equivalent to the existing `wbra_dfhcnv_key` parameter, with the important difference that using the new parameters means you do not have to create entries in the code page conversion table (DFHCNV). You can simply specify the character set and host code page, and CICS determines the appropriate conversion template. If your existing analyzer program uses the `wbra_dfhcnv_key` parameter, then until you change to the new parameters or remove the analyzer program from the processing path for requests, you need to retain the relevant DFHCNV entries for migration purposes.

- A new output parameter `wbra_commarea` is provided to indicate where an application that does not use the **EXEC CICS WEB** API commands requires pre-CICS TS Version 3 compatibility processing. This flag is for existing applications in the specific circumstance where the Web client needs a response that is identical with the response it would have received before CICS TS Version 3. Setting this flag means that:

  - CICS does not add any response headers that would not have been used before CICS TS Version 3.

  - If error processing is required, CICS sends an error response that is suitable for, and labeled as, an HTTP/1.0 response, regardless of the HTTP version of the Web client. CICS would normally reply to a HTTP/1.1 client with an HTTP/1.1 error response, but this might mislead the client into thinking that the application would normally send a response at HTTP/1.1 level.

  A URIMAP definition may be set up for the request, but it must specify the analyzer program.

Analyzer programs cannot be invoked when CICS is an HTTP client, or for Web service processing; they can only be invoked when CICS is an HTTP server.

Changes to invocation of the converter program

Use of the converter program for CICS Web support processing for HTTP requests is still optional. Converter programs are primarily for use with application programs which were not originally coded for use with the Web, and need to receive input in

the form of a COMMAREA. They can be used to convert output from one or more of these application programs into an HTTP message. Web-aware application programs, which are coded using the `EXEC CICS WEB` and `EXEC CICS DOCUMENT` application programming interfaces, should not require this conversion to take place.

The URIMAP definition can specify that a converter program is to carry out relevant processing for HTTP requests. The PROGRAM attribute of the URIMAP definition is passed to the converter program, and the converter program can choose to override it. If an analyzer program is used in CICS Web support processing, the analyzer program can also specify a converter program, and can pass data to the converter program in a COMMAREA or user token. If your existing analyzer program passes data to a converter program in this way, note that this function cannot be replicated by a URIMAP definition.

A converter program is not able to specify code page conversion settings for a request that it receives in a COMMAREA. If a converter program is specified in a URIMAP definition, and the headers for the Web client's request indicate that the message body is text, CICS converts the message body supplied in the COMMAREA using the following standard settings:

- For the character set, if the Web client's request has a Content-Type header naming a character set supported by CICS, that character set is used. If the Web client's request has no Content-Type header or the named character set is unsupported, the ISO-8859-1 character set is used.
- For the host code page, CICS uses the default code page for the local CICS region, as specified in the LOCALCCSID system initialization parameter.

If these standard settings are not suitable, or if code page conversion is *not* wanted, an analyzer program must be used in the processing path to specify alternative settings.

The converter program cannot be invoked when CICS is an HTTP client, or for Web service processing; it can only be invoked when CICS is an HTTP server.

Changes to the Web error program

When a request error or an abend occurs in the CICS Web support process for CICS as an HTTP server, a user-replaceable Web error program provides an error response to the Web client. A Web error program receives or obtains information about the error situation. The program can customize the default HTTP response (including status code and status text) that CICS plans to send to the Web client, or build its own HTTP response, and return it to CICS for sending.

A new user-replaceable Web error transaction program, DFHWBERX, is supplied. DFHWBERX is used when the new CICS-supplied default analyzer DFHWBAAX is specified as the analyzer program on the TCPIPSERVICE definition, and no matching URIMAP definition is found for a request. In this situation, DFHWBAAX specifies DFHWBERX as the application program to handle the request. DFHWBERX could also be specified in another analyzer program, or as the PROGRAM attribute in a URIMAP definition if an error response is always wanted for the request.

DFHWBERX provides error handling as follows:

- If the request is a POST request with media type `text/xml`, it is assumed to be a SOAP 1.1 request, and a SOAP 1.1 fault response is returned.

- If the request is a POST request with media type `application/soap+xml`, it is assumed to be a SOAP 1.2 request, and a SOAP 1.2 fault response is returned.
- All other requests are assumed to be a standard HTTP request, so a suitable HTTP response is composed and returned with a 404 (Not Found) status code.

The **EXEC CICS WEB** and DOCUMENT application programming interfaces are available from DFHWBERX. It does not use information provided in a COMMAREA, but instead uses the **EXEC CICS** commands to obtain information about the Web client's request and create and send the error response.

The Web error program DFHWBEP, which was available before CICS TS 3.1, is used in all other situations where a Web error program is required. You might have customized the Web error program DFHWBEP in an earlier CICS release. CICS now uses additional status codes, and uses some existing status codes in a wider range of situations. You should be aware of this if you have made modifications to DFHWBEP to customize the responses associated with each status code. The **EXEC CICS WEB** and DOCUMENT application programming interfaces are not available from DFHWBEP. DFHWBEP uses a COMMAREA-based interface where a complete HTTP response is created as a buffer of data.

In the COMMAREA passed to a Web error program, there is a new input parameter `wbep_activity`, which specifies the type of processing that was in progress when the error occurred. 0 indicates server processing, and 2 indicates pipeline processing.

Changes to statistics

New global statistics and resource statistics are produced for the URIMAP resource definition object. The global statistics show usage counts for different CICS Web support operations involving URIMAP definitions, such as successful matches and unsuccessful match attempts, redirection, or delivery of a dynamic or static response. The resource statistics show attributes and relevant usage counts for individual URIMAP definitions.

The statistics are collected by the PERFORM STATISTICS and EXTRACT STATISTICS commands, using the URIMAP keyword. The DSECTs are DFHWBRDS (for resource statistics) and DFHWBGDS (for global statistics). DFHSTUP and DFH0STAT include new reports for these statistics.

Changes to CICS utilities
Statistics utility program DFHSTUP

DFHSTUP supports the changes to statistics described in "Changes to statistics."

You can now code the URIMAP resource type on the `SELECT TYPE` and `IGNORE TYPE` control parameters for DFHSTUP.

Changes to problem determination

New messages and trace points, and one new abend code, are introduced for the enhancements to CICS Web support. Any information received by CICS as Warning headers in HTTP messages is collected and written to the TD queue CWBW.

Warning messages in HTTP headers

If the Warning header is present on an HTTP message, it normally contains information that is intended to be read by a user. If CICS Web support receives a message with a Warning header, the text associated with the header, and the IP address of the sender, is written to the transient data queue CWBW. (CWBW is indirected to CSSL.) If you receive too many warning headers, you can remove the CWBW transient data queue to suppress these records.

Messages

New DFHWBxxxx messages are introduced as a result of the enhancements to CICS Web support. All new and changed messages are described in the *CICS Messages and Codes* manual.

Abend codes

One new abend code is introduced, AWBP. It is used when a Web-aware application for CICS as an HTTP server is sending a chunked message, but fails to send the final empty chunk to complete the message.

Trace

New CICS trace points are added to the WB domain trace for the enhancements to CICS Web support. The new trace points are in the range WB 0419 to WB 0C07.

To control the output of CICS trace information, use CICS trace control in the normal way.

Security

Security for new SPI and CEMT commands

New predefined RACF resource names control access to the following resources using the SPI and CEMT:
>   HOST
>   URIMAP

New category 1 transaction

The new CWXU transaction is for CICS internal use, and should not be invoked from a user terminal. For security purposes, it is a category 1 transaction.

New global user exits

When CICS is an HTTP client, the new global user exits XWBOPEN (on the WEB OPEN command) and XWBSNDO (on the WEB SEND command) enable you to apply a security policy to the host name and path specified for outbound HTTP client requests from CICS. "Changes to global user exits" on page 106 describes these new exits.

Security for static responses by CICS as an HTTP server

You can deliver CICS documents and HFS files as static responses to requests from Web clients, by setting up URIMAP definitions that supply the response

without calling a user-written application program. When you deliver items as a
static response, HTTP basic authentication does not operate. This means that
resource level security, with access controls based on a user ID, cannot be applied
to items delivered as a static response. If the items require authentication or
resource level security, you need to deliver the material as an application-generated
response. When an application-generated response is used, basic authentication
can be used, and the user ID from basic authentication can be applied to the alias
transaction that covers processing by the user-written application program, so you
can grant or deny access to the specific resources and commands used by the
application program.

Migration

Migration of existing CICS Web support applications

CICS Transaction Server for z/OS, Version 3 Release 1 is designed to support your
existing CICS Web support architecture for both Web-aware and non-Web-aware
application programs. The **EXEC CICS WEB** API command changes are designed to
allow existing Web-aware application programs that send and receive HTTP
messages to work unchanged, until you choose to migrate them to take advantage
of the enhancements that are now available. If you continue to use existing CICS
Web support applications, note these migration points:

* **If you are using CICS Web support to process non-HTTP requests, specify
  the new USER protocol on the TCPIPSERVICE definition that defines the
  port for these requests.** This also applies to HTTP requests with nonstandard
  request methods, which are now rejected if they are received on the HTTP
  protocol (previously, they were accepted and processed as non-HTTP).
  Processing for all non-HTTP requests must now be carried out under the USER
  protocol, so that they are not subjected to the basic acceptance checks which
  CICS carries out for requests using the HTTP protocol. The requests are flagged
  as non-HTTP and passed unchanged to the analyzer program for the
  TCPIPSERVICE. CICS Web support facilities are used for handling the request,
  but no acceptance checks are carried out for messages sent and received using
  this protocol.

  Note: Because only one active TCPIPSERVICE definition can exist for each
  port, non-HTTP requests can no longer use the same port as HTTP
  requests. The well-known port numbers 80 (for HTTP) and 443 (for
  HTTPS) must have the HTTP protocol and therefore cannot accept
  non-HTTP requests. Web clients must specify any changed port in the
  URL for their requests.

* **Check the settings for your TCPIPSERVICE resource definitions with the
  HTTP protocol.**

  1. The SOCKETCLOSE attribute must no longer have a zero setting
     (SOCKETCLOSE(0)). A zero setting for SOCKETCLOSE means that CICS
     closes the connection immediately after receiving data from the Web client,
     unless further data is waiting. This means that persistent connections cannot
     be maintained. A non-zero setting for SOCKETCLOSE enables persistent
     connections with both HTTP/1.1 clients, and HTTP/1.0 clients (where the
     client supports this).

  2. The new MAXDATALEN option should be specified to limit the maximum
     length of data that may be received by CICS as an HTTP server. This setting
     helps to guard against denial of service attacks involving the transmission of
     large amounts of data.

3. If you are using SSL, there are some changes to the security options available on the TCPIPSERVICE resource definition.

- **The analyzer program now allows you to supply codepage conversion parameters to CICS Web Support instead of supplying the name of a DFHCNV table entry.** If you want to continue to use an analyzer program that you coded in an earlier CICS release to reference DFHCNV, you must either continue to supply the entries in the code page conversion table, or change the analyzer program. Changing the analyzer program involves coding two new output parameters to specify the client and server code pages, in place of the output parameter that specified the name of a DFHCNV entry. If you do this, you do not need to migrate your DFHCNV entries.

- **If you have modified the user-replaceable Web error program DFHWBEP to customize the HTTP responses provided in error situations, be aware that CICS now uses additional status codes, and uses some existing status codes in a wider range of situations.**

  1. Check that your program is using an appropriate range of input parameters to identify the situation to which the customized response applies, rather than relying on the status code alone. The error code, abend code, message number, response and reason codes, or program name can be used to identify the situation that has given rise to the HTTP response. If these checks are not made, you might find that where CICS is using the status code for a new purpose, an inappropriately customized response is returned.

  2. Check that your program includes logic to pass through unchanged any HTTP responses with status codes that are not known to the program.

- **The DFHWBCLI interface is still supported in CICS Transaction Server for z/OS, Version 3 Release 1.** To gain enhanced functionality, you can migrate HTTP client applications that used the DFHWBCLI interface, to use the `EXEC CICS WEB` API commands for client requests (with the SESSTOKEN option). One important difference to note is that in the `EXEC CICS WEB` API, the use of a proxy server is specified by a user exit on the WEB OPEN command (XWBOPEN), and the URL of the proxy server is supplied by that user exit. Chapter 3, "Support for HTTP client requests from CICS applications," on page 83 describes how HTTP client requests can now be made.

Migration to the new CICS Web support function

CICS Web support in CICS Transaction Server for z/OS, Version 3 Release 1 has many enhancements to provide automatic and administrator control of functions that were previously handled by user-replaceable programs. In particular, you are recommended to investigate migration possibilities for the following elements of your CICS Web support architecture:

- You should usually be able to replace the request processing functions of an existing analyzer program with URIMAP resource definitions, which can be changed and controlled using CICS system programming commands. URIMAP definitions can be used to match the URLs of requests and map them to application programs, and specify a converter program and alias transaction. If your analyzer program is customized to provide additional functions, you can continue to use it instead of a URIMAP definition, or you can combine it with a URIMAP definition. While migrating to the use of URIMAPs:

  1. You can introduce URIMAP resource definitions progressively for a small number of requests at a time. Depending on the type of processing carried out by your analyzer program, and the type of application that handles the request, you can choose whether or not to continue using the analyzer program in the processing path for each request.

2. You might prefer to select and publish new URLs for requests handled by URIMAP resource definitions, rather than retaining your existing URLs. When you are ready to discontinue the use of the old processing path for a request, you can set up a URIMAP definition to permanently redirect requests from the old URL to the new URL.

3. Ensure that the analyzer program specified on the TCPIPSERVICE definition still contains basic error handling procedures, even if it is no longer involved in the processing path for requests. The analyzer program must still be present, and it receives requests if URIMAP matching fails.

- For application programs that do not use the **EXEC CICS WEB** API commands but produce an HTTP response in a COMMAREA, CICS Web support is not able to assist with assembling the message structure correctly, or to carry out its full range of checks on the response. To take advantage of all the available CICS Web support facilities, it is recommended that you plan to convert these applications to Web-aware application programs that use the WEB API commands.

- URIMAP resource definitions can be used to deliver the contents of a CICS document or HFS file as a static response, or to deliver a redirection response, without involving a user-written application program. You could consider using this mechanism, instead of an application program, for simple responses that do not involve dynamic processing.

- Check that code page conversion is operating in the most efficient way. With minor changes to your application, you can take advantage of new CICS Web support facilities to:
  – Avoid setting up and using a code page conversion table (DFHCNV) for CICS Web support.
  – Allow CICS to identify and use the Web client's character set for code page conversion, rather than specifying this yourself.
  – Use the local system default (LOCALCCSID system initialization parameter) to identify the application program's code page, rather than specifying this yourself.
  – Convert to and from the UTF-8 and UTF-16 character sets.

CICSPlex SM support

The CPSM Web User Interface (WUI) is accessed by Web browsers. The interface accepts both HTTP/1.0 and HTTP/1.1 level requests, but provides only HTTP/1.0 responses. Web browsers that are used to access the CPSM WUI should be able to accept and understand HTTP/1.0 responses.

Changes to CICSPlex SM end user interface views

Changes have been made to the following EUI views:
- "TCPDEF view"
- "DOCDEF view" on page 170
- "DOCTEMP view" on page 170

TCPDEF view

The following attributes have been added to the **TCPDEF** view:

PROTOCOL CVDA (USER)
> Protocol

MAXDATALEN
　　　　Defines the maximum length of data that can be received on an inbound
　　　　request or a response to an outbound request.

The **PRIVACY** attribute in this view is no longer valid in CICS Transaction Server
3.1 and will be ignored.

DOCTEMP view

The following attributes have been added to the **DOCTEMP** view:

HFSFILE
　　　　A UNIX System Services HFS file to be used as a document template.

TEMPLATETYPE CVDA (HFS)
　　　　Document template type

DOCDEF view

A new attribute has been added to the **DOCDEF** view:

HFSFILE
　　　　A UNIX System Services HFS file to be used as a document template.

The attribute **TEMPLATETYPE** has been removed.

Changes to CICSPlex SM application programming interface
New resource tables

The following have been introduced:

URIMAP resource table

The URIMAP resource table has the following SPI attributes:

URIMAP
　　　　URIMAP name

ENABLESTATUS
　　　　Shows the CVDA value that indicates whether the map is to be installed in
　　　　ENABLED or DISABLED state.

USAGE
　　　　Shows the CVDA value of Server, Client, or Pipeline indicating whether the
　　　　map is respectively for CICS as an HTTP server, CICS as an HTTP client,
　　　　or Web services.

SCHEME
　　　　Shows the CVDA value which indicates whether the scheme component of
　　　　the URI to which the URI map applies is without SSL (HTTP) or with SSL
　　　　(HTTPS).

HOST　Shows the 116-character host component of the URI to which the map
　　　　applies.

PATH Shows the 255-character path component of the URI to which the map applies.

MEDIATYPE
Shows the 56-character data content of the static response that CICS provides to the inbound HTTP request.

CHARACTERSET
Shows the 40-character name of the character set to which CICS converts the static response that is sent to the inbound HTTP request.

HOSTCODEPAGE
Shows the 10-character name of the IBM codepage (EBCDIC) in which the text document that forms the static response is encoded.

TEMPLATENAME
Shows the 48-character name of a CICS document template that forms the static response.

HFSFILE
Shows the 255-character fully-qualified name of the UNIX system services HFS file which forms the static response sent to an inbound HTTP request.

TCPIPSERVICE
Shows the 8-character name of the TCPIPSERVICE resource definition that defines the inbound port to which the URI map applies.

ANALYZER
Shows the CVDA value (YES or NO) which shows whether or not an analyzer program is to be involved in processing the inbound HTTP request.

CONVERTER
Shows the 8-character name of a converter program which is to process the content of the request.

TRANSACTION
Shows the 4-character name of an alias transaction used to run the user application that composes a response.

PROGRAM
Shows the 8-character name of the user application program or XML service that specifies the service that deals with the request.

PIPELINE
Shows the 8-character name of the Pipeline that specifies the service that deals with the request.

WEBSERVICE
Shows the 32-character name of the Web service that specifies the service that deals with the request.

USERID
Shows the 8-character user-ID used to attach the alias transaction.

CERTIFICATE
Shows the 32-character label of the certificate to be used as the SSL client certificate when the URI is used for an outbound HTTPS request.

CIPHERS
Shows a string of up to 56 hexadecimal digits that is interpreted as a list of up to 28 2-digit cipher suite codes used for outbound SSL requests.

NUMCIPHERS
Number of SSL cipher suite codes

LOCATION
Shows the 255-character URI to which the inbound HTTP request should be redirected.

REDIRECTTYPE
Shows the CVDA value (NONE, TEMPORARY or PERMANENT) for the type of redirection.

MAPREFCOUNT
URI map reference count

MATCHDISABLD
URI map host or path disabled

MATCHREDIREC
URI map host or path redirect

URIMPDEF resource table

The URIMPDEF resource table includes the following RDO attributes:

URIMAP
URI map definition name

STATUS
Shows the CVDA value that indicates whether the map is to be installed in ENABLED or DISABLED state.

USAGE
Shows the CVDA value of Server, Client, or Pipeline indicating whether the map is respectively for CICS as an HTTP server, CICS as an HTTP client, or Web services.

SCHEME
Shows the CVDA value which indicates whether the scheme component of the URI to which the URI map applies is without SSL (HTTP) or with SSL (HTTPS).

HOST Shows the 116-character host component of the URI to which the map applies.

PATH Shows the 255-character path component of the URI to which the map applies.

MEDIATYPE
Shows the 40-character data content of the static response that CICS provides to the inbound HTTP request.

CHARACTERSET
Shows the 40-character name of the character set to which CICS converts the static response that is sent to the inbound HTTP request.

HOSTCODEPAGE
Shows the 10-character name of the IBM codepage (EBCDIC) in which the text document that forms the static response is encoded.

TEMPLATENAME
Shows the 48-character name of a CICS document template that forms the static response.

HFSFILE
Shows the 255-character fully-qualified name of the UNIX system services HFS file which forms the static response sent to an inbound HTTP request.

TCPIPSERVICE
Shows the 8-character name of the TCPIPSERVICE resource definition that defines the inbound port to which the URI map applies.

ANALYZER
Shows the CVDA value (YES or NO) which shows whether or not an analyzer program is to be involved in processing the inbound HTTP request.

CONVERTER
Shows the 8-character name of a converter program which is to process the content of the request.

TRANSACTION
Shows the 4-character name of an alias transaction used to run the user application that composes a response.

PROGRAM
Shows the 8-character name of the user application program or XML service that specifies the service that deals with the request.

PIPELINE
Shows the 8-character name of the Pipeline that specifies the service that deals with the request.

WEBSERVICE
Shows the 32-character name of the Web service that specifies the service that deals with the request.

USERID
Shows the 8-character user ID used to attach the alias transaction.

CERTIFICATE
Shows the 32-character certificate label to be used as the SSL client certificate when the URI is used for an outbound HTTPS request.

CIPHERS
Shows a string of up to 56 hexadecimal digits that is interpreted as a list of up to 28 2-digit cipher suite codes used for outbound SSL requests.

LOCATION
Shows the 255-character URI to which the inbound HTTP request should be redirected.

REDIRECTTYPE
Shows the CVDA value (NONE, TEMPORARY or PERMANENT) for the type of redirection.

URIMPGBL resource table

The URIMPGBL resource table includes the following attributes:

MAPREFCOUNT
URI map reference count

MATCHDISABLD
URI map host or path disabled

NOMATCHCOUNT
URI map host or path no match count

MATCHCOUNT
URI map host or path match count

MATCHREDIREC
URI map host or path redirect

MATCHANALYZE
URI map host or path match analyzer

STATICONTENT
URI map static content

DYNAMCONTENT
URI map dynamic content

PIPELINEREQS
URI map pipeline requests

SCHEMEHTTP
URI map scheme (HTTP) requests

SCHEMEHTTPS
URI map scheme (HTTPS) requests

HOSTDISABLED
Host disabled count

HOST resource table

The HOST resource table has the following attributes:

HOSTNAME
Host name

ENABLESTATUS
Shows the CVDA value that indicates whether the host is to be installed in ENABLED or DISABLED state.

Changes to resource tables

Changes have been made to the following:
* "TCPDEF resource table"
* "DOCTEMP resource table" on page 175
* "DOCDEF resource table" on page 175
* "TASK resource table" on page 175

TCPDEF resource table

The TCPDEF resource table has the following additional attributes:

PROTOCOL CVDA (USER)
Protocol

MAXDATALEN
Defines the maximum length of data that may be received or sent

\# The **PRIVACY** attribute in this resource table is no longer valid in CICS Transaction
\# Server 3.1 (or later versions) and will be ignored.

DOCTEMP resource table

The DOCTEMP resource table has the following additional attributes:

HFSFILE
UNIX Sytem Service Hierarchical File System template file

TEMPLATETYPE
Document template type

DOCDEF resource table

The DOCDEF resource table includes the following attribute:

HFSFILE
Hierarchical File System template file

TASK resource table

The TASK resource table includes the following attributes:

TMRWBRDL
Shows the data length of data read from the repository

TMRWBWDL
Shows the data length of data written to the repository

Changes to CICSPlex SM Web User Interface

New WUI views

The following WUI views have been introduced:
- "URI mapping definitions view"
- "URI map view" on page 176
- "URI map global view" on page 176
- "Host view" on page 176

URI mapping definitions view

A definitional view set has been introduced called **URI mapping definitions**, associated with the new URIMPDEF resource table. The view name for this tabular view is EYUSTARTURIMPDEF.TABULAR and the existing EYUSTARTADMRES menu has been extended to include it.

To open the **URI mapping definitions** view, do the following:
1. Click **Administration views** from the Main menu
2. Click **Basic CICS resource administration views** (or, alternatively, click **Fully functional Business Application Services (BAS) administration views**)
3. Click **CICS resource definitions**
4. Scroll down and click **URI mapping definitions**

The **URI mapping definition** view is displayed This view includes the following five action buttons:
- **Create**
- **Update**
- **Remove**

- **Install**
- **Add to Resource group**

See the attributes of the URIMPDEF resource table listed in"URIMPDEF resource table" on page 172 for field details.

URI map view

A new view has been introduced called **URI map**, associated with the new URIMAP resource table. The view name for this tabular view is EYUSTARTURIMAP.TABULAR and the existing EYUSTARTTCPIPS menu has been extended to include it.

To open the **URI map** view, do the following:

1. Click **CICS operations view** from the Main menu
2. Scroll down and click **TCP/IP service operations views**
3. Click **URI map**

The **URI map** view is displayed. This view includes the following four action buttons:

- **Set attributes**
- **Enable**
- **Disable**
- **Discard**

For attribute details of the URIMAP resource table see "URIMAP resource table" on page 170

URI map global view

A new view has been introduced called **URI map global**, associated with the new URIMPGBL resource table. The view name for this tabular view is EYUSTARTURIMPGBL.DETAILED and the existing EYUSTARTTCPIPS menu has been extended to include it.

To open the **URI map global** view, do the following:

1. Click **CICS operations view** from the Main menu
2. Scroll down and click **TCP/IP service operations views**
3. Click **URI map global**
4. Select a **CICS system name** to open the **URI map global** detailed view

The **URI map global** view is displayed. For attribute details of the URIMPGBL resource table see "URIMPGBL resource table" on page 173.

Host view

A new view has been introduced called **Host**, associated with the new HOST resource table. The view name for this tabular view is EYUSTARTHOST.TABULAR and the existing EYUSTARTTCPIPS menu has been extended to include it.

To open the **Host** view, do the following:

1. Click **CICS operations view** from the Main menu
2. Scroll down and click **TCP/IP service operations views**
3. Click **Host**

The **Host** view is displayed.

For attribute details of the HOST resource table see "HOST resource table" on page 174.

Changes to the WUI views

Changes have been made to the following views:
- "TCP/IP Service definition view"
- "Document template definition view"
- "Document template view"
- "TCP/IP usage view"

TCP/IP Service definition view

The following attributes have been added to the TCPDEF (EYUSTARTTCPDEF) view:

PROTOCOL CVDA (USER)
> Protocol

MAXDATALEN
> Defines the maximum length of data that can be received on an inbound request or a response to an outbound request.

The **PRIVACY** attribute in this view is no longer valid in CICS Transaction Server 3.1 and will be ignored.

Document template definition view

The following attribute has been added to the DOCDEF (EYUSTARTDOCDEF.DETAILED) view:

HFSFILE
> A UNIX System Services HFS file to be used as a document template.

Document template view

The following attributes have been added to the DOCTEMP (EYUSTARTDOCTEMP.DETAILED) view:

HFSFILE
> A UNIX System Services HFS file to be used as a document template.

TEMPLATETYPE CVDA (HFS)
> Document template type

TCP/IP usage view

The following attributes have been added to the **TCP/IP usage** view in the Active task view set (EYUSTARTTASK.DETAIL7), within the Task operations view:

TMRWBRDL
> Shows the data length of data read from the repository

TMRWBWDL
> Shows the data length of data written to the repository

Chapter 6. Improvements to Internet security

An overview of the enhancements to security.

The improvements to security cover a variety of areas and are summarized as follows:

- Support for the TLS 1.0 protocol and AES cipher suites
- Scalability through an increased number of simultaneous SSL connections
- Support for certificate revocation lists
- SSL caching across CICS regions in a sysplex
- Specifying minimum as well as maximum encryption negotiation levels

Note: CICS supports both SSL and TLS security protocols in this release. For clarity, the term SSL is used to refer to both protocols in the documentation, except where a specific point about either protocol is required.

Benefits of improvements to Internet security

There are a range of benefits that come from the improvements to security.

CICS now supports the Transport Layer Security (TLS) 1.0 protocol as well as SSL 3.0, allowing you to use the new AES cipher suites that offer 128-bit and 256-bit encryption.

There are improvements to the performance of SSL to support new functions such as Web Services. The number of simultaneous SSL connections that can be used in the system at one time has increased to achieve better throughput.

There is more flexibility in controlling the encryption negotiation between client and server. You can specify a minimum as well as a maximum encryption level in CICS for negotiating with particular users.

CICS can now check all certificates against a certificate revocation list (CRL) when negotiating with clients. Any connections using revoked certificates are closed immediately.

You can specify whether you want to share session IDs across a sysplex by using the SSL cache. CICS performs a partial SSL handshake if the client has negotiated with CICS previously. Sharing the cache across a number of CICS regions improves the performance of SSL negotiation and connection throughput.

Security terminology

The following terminology is used to describe the enhancements to Internet security in CICS.

Transport Layer Security (TLS)
  A security protocol that is used to provide secure communication over the Internet. The specification is documented in RFC2246.

cipher suite
  A combination of an encryption algorithm, encryption key length and MAC algorithm that is negotiated during an SSL handshake.

Message Authentication Code (MAC)
A cryptographically secure hash code that is associated with each message sent over an SSL connection.

MAC algorithm
A cryptographic algorithm that calculates a message authentication code. SSL uses the MD5 and SHA algorithms.

SSL cache
The cache that is used by SSL to store session id information about its encryption negotiation with clients. If a client has previously securely connected to CICS using SSL, only a partial handshake is performed to establish the SSL connection.

certificate revocation list
A list of revoked certificates that is provided by independent bodies called certificate authorities. If a certificate has been withdrawn, it is added to a certificate revocation list. These lists can be cross-referenced during the SSL handshake negotiation when the client and server try to authenticate one another.

SP mode
The TCB mode that owns the initial pthread-owning task. The initial pthread-owning task owns all the pthreads that are used by S8 TCBs.

SSL pool
The pool that contains and manages the S8 TCBs in a CICS region.

SSL handshake
An exchange of information that takes place between a client and server when a connection is established. The handshake involves the negotiation of which encryption algorithms to use, and authentication of one another.

Requirements

Hardware

zSeries® cryptographic hardware is required to fully benefit from the performance improvements to SSL encryption, which relies upon the z/OS Integrated Cryptographic Facility (ICSF).

Software

The System SSL Security Level 3 feature for z/OS is required to use cipher suites with 128-bit encryption or above.

Transport Layer Security protocol

The Transport Layer Security (TLS) 1.0 protocol is a standard security protocol, that provides secure communication over the Internet.

CICS supports two security protocols that can be used to provide secure communication over the Internet. The first is the Secure Sockets Layer (SSL) 3.0 protocol. The second is the Transport Layer Security (TLS) 1.0 protocol, which is the latest industry standard SSL protocol and is based on SSL 3.0. The TLS 1.0 specification is documented in RFC2246 and is available on the Internet at www.rfc-editor.org/rfcsearch.html. Any connections that require encryption will automatically use the TLS protocol, unless the client specifically requires SSL 3.0.

The primary aim of TLS is to make the Secure Sockets Layer more secure and to make the specification of the protocol more precise and complete. TLS provides the following enhancements over SSL 3.0:

Key-Hashing for Message Authentication
> TLS uses Key-Hashing for Message Authentication Code (HMAC), which ensures that a record cannot be altered while travelling over an open network such as the Internet. SSL Version 3.0 also provides keyed message authentication, but HMAC is considered more secure than the (Message Authentication Code) MAC function that SSL Version 3.0 uses.

Enhanced Pseudorandom Function (PRF)
> PRF is used for generating key data. In TLS, the PRF is defined with the HMAC. The PRF uses two hash algorithms in a way that guarantees its security. If either algorithm is exposed then the data remains secure as long as the second algorithm is not exposed.

Improved finished message verification
> Both TLS 1.0 and SSL 3.0 provide a finished message to both endpoints that authenticates that the exchanged messages were not altered. However, TLS bases this finished message on the PRF and HMAC values, which is more secure than SSL Version 3.0.

Consistent certificate handling
> Unlike SSL 3.0, TLS attempts specify the type of certificate which must be exchanged between TLS implementations.

Specific alert messages
> TLS provides more specific and additional alerts to indicate problems that either session endpoint detects. TLS also documents when certain alerts should be sent.

Improvements to SSL performance

There are changes to the implementation of S8 TCBs to improve the number and performance of SSL connections in CICS.

CICS uses the open transaction environment (OTE) to manage SSL connections. Each SSL connection uses an S8 TCB, which now runs as a UNIX pthread. There is also a new open TCB mode called SP, that is used for socket pthread owning tasks. All of the S8 TCBs run within a single enclave, which is owned by the SP TCB and also contains the SSL cache. This provides the benefit of saving storage below the line, allowing many more simultaneous SSL connections in CICS than previous releases.

The S8 TCBs are contained in an SSL pool, which is managed by the CICS dispatcher. The S8 TCBS are allocated from the new SSL pool, but are only locked to a transaction for the period that it needs to perform SSL functions. After the SSL negotiation is complete, the TCB is released back into the SSL pool to be reused. The MAXSSLTCBS system initialization parameter specifies the maximum number of S8 open TCBs in the SSL pool. The default value is 8, but you can specify up to 1024.

You can monitor the performance of the SSL pool and the S8 TCBs using the dispatcher reports from DFH0STAT and DFHSTUP. The statistics include information on how often the maximum number of S8 TCBs are reached, the delay before a TCB is allocated and the actual number of TCBs in the SSL pool.

Using certificate revocation lists

You can configure CICS to use certificate revocation lists (CRLs) to check the validity of client certificates being used in SSL negotiations.

To use certificate revocation lists, you must install and configure an LDAP server. Details on how to perform these tasks can be found in *z/OS V1R4.0 Security Server LDAP Server Admin and Use*.

Certificate revocation lists are available from certificate authorities such as Verisign. They are kept in CRL repositories that are available on the world wide web and can be downloaded and stored in an LDAP server. To populate the LDAP server and update certificate revocation lists, use the CICS-supplied transaction CCRL. You can run the CCRL transaction from a terminal or using a START command. To include CRLs in your LDAP server, follow these steps:

1. Configure the LDAP server to specify which certificate authorities you want to use.

\# 2. Specify the name of the RACF profile that authorizes CICS to access the CRLs
\# in the LDAP server using the CRLPROFILE system initialization parameter.

3. Run the CCRL transaction.

The SSL cache

The SSL cache is used to store session ids from the negotiation between clients and CICS.

The SSL cache allows CICS to perform partial handshakes with clients that it has previously authenticated. In a local CICS region, the SSL cache is part of the enclave for the S8 TCBs. You have the option of sharing the SSL cache across a sysplex if this is appropriate for your CICS system. You can use sysplex caching if you have multiple CICS socket-owning regions that accept SSL connections at the same IP address.

If you want to share the cache between regions, activate the system SSL started task GSKSRVR and use the system initialization parameter SSLCACHE. The default is to use the local region cache, but you can change this by specifying the option SYSPLEX. CICS will use the SSL cache in the coupling facility instead to store session ids.

Customizing encryption negotiations

You can select the cipher suites that are used in the encryption negotiation process to set a minimum level as well as a maximum level of encryption.

The CIPHERS attribute on the resource definitions TCPIPSERVICE, CORBASERVER, and URIMAP specifies the cipher suites that can be used for each encryption level. The default value of the attribute is the list of 2-digit cipher codes that are used in encryption negotiations. You have the option of customizing this list of cipher suites to include your order of preference for the encryption levels at which CICS should negotiate with clients. You can also choose to remove cipher suites from the list. This is particularly useful if you want to ensure that only a very high level of encryption is used. You can do this as follows:

1. Select the resource definition that you want to change.

2. The CIPHERS attribute displays the default value. For example, if the system initialization parameter ENCRYPTION=STRONG, the default value in z/OS 1.9 is `050435363738392F303132330A1613100D0915120F0C03060201`.

3. Edit the attribute value to remove and reorder the cipher suites. For example, you could specify `352F0A0504`.

4. Save the resource definition.

Specifying `352F0A0504` means that CICS will not negotiate below 128-bit encryption for connections using this resource. Each of the 2-digit codes in the attribute, for example 35, 2F, 0A and so on, refer to cipher suites that have at least a 128-bit encryption. CICS will start by trying to negotiate using the AES cipher suites 35 and 2F, because these are first in the list of cipher codes. If the client does not have this level of encryption, CICS will close the connection.

Note that you cannot include cipher suites that are not in the default values for that level of encryption. For example, if you have a MEDIUM level of encryption specified, you cannot add the AES cipher suites 35 and 2F to the CIPHERS attribute.

Changes to CICS externals

Changes to system initialization parameters

There are new and changed system initialization parameters for the improvements to Internet security. The changed parameters are:

ENCRYPTION={STRONG|WEAK|MEDIUM}
Specifies the cipher suites that CICS uses for secure TCP/IP connections. For compatibility with previous releases, ENCRYPTION=NORMAL is accepted as an equivalent to ENCRYPTION=MEDIUM.

STRONG
Specifies that CICS should use only the following cipher suites:

Cipher suite	Encryption algorithm	Key length	MAC algorithm
01	No encryption		MD5
02	No encryption		SHA
03	RC4	40 bits	MD5
04	RC4	128 bits	MD5
05	RC4	128 bits	SHA
06	RC2	40 bits	MD5
09	DES	56 bits	SHA
0A	Triple DES	168 bits	SHA
2F	AES	128 bits	SHA
35	AES	256 bits	SHA

Cipher suite	Encryption algorithm	Key length	MAC algorithm
The terms used in this table are:			
MD5 Message Digest algorithm			
SHA Secure Hash algorithm			
RC2, RC4 Rivest encryption			
DES Data Encryption Standard			
Triple DES DES applied three times			
AES Advanced Encryption Standard			

WEAK

Specifies that CICS should use only the following cipher suites:

Cipher suite	Encryption algorithm	Key length	MAC algorithm
01	No encryption		MD5
02	No encryption		SHA
03	RC4	40 bits	MD5
06	RC2	40 bits	MD5
The terms used in this table are:			
MD5 Message Digest algorithm			
SHA Secure Hash algorithm			
RC2, RC4 Rivest encryption			

MEDIUM

Specifies that CICS should use only the following cipher suites:

Cipher suite	Encryption algorithm	Key length	MAC algorithm
01	No encryption		MD5
02	No encryption		SHA
03	RC4	40 bits	MD5
06	RC2	40 bits	MD5
09	DES	56 bits	SHA
The terms used in this table are:			
MD5 Message Digest algorithm			
SHA Secure Hash algorithm			
RC2, RC4 Rivest encryption			
DES Data Encryption Standard			

The parameter SSLTCBS is obsolete. Use the following new parameter instead:

MAXSSLTCBS={8|*number***}**
> Specifies the maximum number of S8 TCBs that can run in the SSL pool. The default is 8, but you can specify up to 1024 TCBs.

The new system initialization parameters are:

CRLPROFILE=*profilename*
> Specifies the name of the RACF profile that CICS should use to access the
> LDAP server that contains certificate revocation lists (CRLs). Specifying this
> parameter means that CICS checks each client certificate during the SSL
> negotiation for a revoked status. If the certificate is revoked, CICS closes the
> connection immediately.

SSLCACHE={CICS|SYSPLEX}
> Specifies whether SSL is to use the local or sysplex caching of session ids. Sysplex caching is useful where multiple CICS socket-owning regions accept SSL connections at the same IP address.

Changes to resource definition
TCPIPSERVICE and CORBASERVER definitions

A new attribute, CIPHERS, has been added to the TCPIPSERVICE and CORBASERVER resource definitions. It can also be specified in the new URIMAP resource definition. See URIMAP resource definitions for more information. The CIPHERS list of cipher suite codes is only used when the sockets connection that is established for the resource uses the SSL or TLS security protocols. For a TCPIPSERVICE definition, the CIPHERS list is used for inbound socket connections. For a CORBASERVER definition, the CIPHERS list is used for outbound socket connections.

CIPHERS=*value*
> The value specifies a string of up to 56 hexadecimal digits that is interpreted as a list of up to 28 2-digit cipher suite codes. The list of acceptable codes is dependent on the ENCRYPTION system initialization parameter.
> - For ENCRYPTION=WEAK, the default value is `03060102`
> - For ENCRYPTION=MEDIUM, the default value is `0903060102`
> - For ENCRYPTION=STRONG, the default value is `0504352F0A0903060201`

> You can reorder the cipher codes or remove them from the default list. However, you cannot add cipher codes that are not in the default list for the specified encryption level. The ENCRYPTION system initialization parameter determines the cipher suite codes that are allowed for each encryption level.

> The PRIVACY attribute of the TCPIPSERVICE resource definition reflects the CIPHERS attribute value. Since the default value of the CIPHERS attribute is the complete list of cipher suites, removing some of the cipher codes can change the PRIVACY attribute.
> - If you remove cipher suites 01 and 02 to specify that CICS should only negotiate with clients that have encryption, the PRIVACY attribute value changes to REQUIRED.
> - If you remove all of the cipher suites except cipher suites 01 and 02 to specify that CICS should only negotiate with clients that have no encryption, the PRIVACY attribute changes to NOTSUPPORTED.
> - If you have any other combination of cipher suites specified, including the default, the PRIVACY attribute value is SUPPORTED.

Similar constraints apply to the OUTPRIVACY attribute of the CORBASERVER resource definition.

Changes to the system programming interface

The **EXEC CICS** commands **INQUIRE TCPIPSERVICE**, **INQUIRE CORBASERVER** and the new command **INQUIRE URIMAP** now include two security options.

CIPHERS(*char56*)
Returns the list of cipher suites that is specified in the attribute CIPHERS for
the resource definitions TCPIPSERVICE, CORBASERVER and URIMAP. This
list of cipher suites are used to negotiate SSL connections. For example, if you
were using weak encryption, the default value would be 03060102.

NUMCIPHERS(*halfword*)
Returns the number of cipher suites that are used to negotiate encryption levels as part of the SSL handshake.

The **EXEC CICS** command **INQUIRE TCPIP** also has two security options.

CRLPROFILE(*char246*)
Returns the name of the RACF profile that is specified in the CRLPROFILE
system initialization parameter. The RACF profile contains the LDAP server
name, userid and password that CICS should use to access the certificate
revocation lists.

SSLCACHE(*cvda*)
returns a CVDA value indicating which cache is being used by SSL to store session ids. CVDA values are:

CICS
The local SSL cache for the CICS region is being used by SSL

SYSPLEX
The SSL cache in the coupling facility is being used by SSL.

There are changes to the **INQUIRE DISPATCHER** and **SET DISPATCHER** commands to handle the SSL TCB pool. The following two options have been added:

MAXSSLTCBS(*fullword*)
returns the maximum number of S8 TCBs allowed in the SSL pool, as specified in the MAXSSLTCBS system initialization parameter.

ACTSSLTCBS(*fullword*)
returns the actual number of S8 TCBs in the SSL pool.

Changes to CICS-supplied transactions
CCRL transaction

Use the CCRL transaction to create and update the certificate revocation lists (CRLs) that are stored in an LDAP server. You only need to use CCRL if you are implementing SSL in your CICS regions and want each connection checked for a revoked certificate during the SSL handshake.

The CCRL transaction specifies the location of CRL repositories on the world wide web. CICS downloads the lists from the CRL repository at the specified URL and stores it in the LDAP server. You can specify more than one URL if you need to access multiple CRL repositories.

Before you run the CCRL transaction, you must have the following set up in CICS:

- An LDAP server that is set up and configured to store the certificate revocation lists. .
- The CRLPROFILE system initialization parameter is defined with the name of the
RACF profile that specifies information for accessing the certificate revocation
lists on the LDAP server.

You can run the CCRL transaction from a terminal or from a START command. If you want to schedule regular updates, use the START command option.

To run the transaction from a terminal, ensure that your terminal accepts mixed case characters. Enter the following command: `CCRL url-list` where *url-list* is a space-delimited list of URLs that contain the certificate revocation lists that you want to download. You will then be prompted to provide the administrator distinguished name and password for the LDAP server. This allows CICS to update the LDAP server with the downloaded CRLs.

To run the transaction from a START command, using the following syntax:

```
EXEC CICS START TRANSID(CCRL)
 FROM('admin://adminDN:adminPW url-list')
 LENGTH(url-list-length)
 [INTERVAL(hhmmss)|TIME(hhmmss)]
```

where *adminDN:adminPW* is the distinguished name and password of the LDAP server,*url-list* is a space-delimited list of URLs that contain the certificate revocation lists that you want to download, *url-list-length* is the length of the URL list including the LDAP admin distinguished name and password, and *hhmmss* is the interval or expiration time at which the CCRL transaction is scheduled to run.

For example, you could specify:

```
EXEC CICS START TRANSID(CCRL)
 FROM('admin://cn=ldapadmin:cics31ldap http://crl.CertificateAuthority.com/CRLList1.crl
       http://crl.CertificateAuthority.com/CRLList2.crl')
  LENGTH(132) INTERVAL(960000)
```

This would schedule the CCRL transaction to run in 96 hours and download certificate revocation lists from two specified URLs.

Changes to CEMT

There are changes to the following commands:

- **INQUIRE TCPIPSERVICE**
- **INQUIRE CORBASERVER**
- **INQUIRE TCPIP**
- **INQUIRE DISPATCHER**
- **SET DISPATCHER**

CICS Supplied Transactions has information about these changed commands.

Changes to global user exits

Within the global user exit parameter list, the TCB indicators list now includes an entry for the SP TCB mode. For a full list see the Customization Guide.

Changes to monitoring

New monitoring fields have been added to monitor the MAXSSLTCBS delay time and the Change mode delay time.

DFHTASK

247 (TYPE-S, 'DSCHMDLY', 8 BYTES)

The elapsed time the user task waited for redispatch after a CICS Dispatcher change-TCB mode request was issued by or on behalf of the user task. For example, a change-TCB mode request from a CICS L8 or S8 mode TCB back to the CICS QR mode TCB may have to wait for the QR TCB because another task is currently dispatched on the QR TCB.

Note: This field is a component of the task suspend time, SUSPTIME (014), field.

281 (TYPE-S, 'MAXSTDLY', 8 BYTES)

The elapsed time in which the user task waited to obtain a CICS SSL TCB (S8 mode), because the CICS system had reached the limit set by the system initialization parameter MAXSSLTCBS. The S8 mode open TCBs are used exclusively by SSL pthreads.

Note: This field is a component of the task suspend time, SUSPTIME (014), field.

Changes to statistics

The changes to statistics include reporting on the new SP TCB mode and the new SSL TCB pool. There are also changes to the statistics utility program DFHSTUP and to the statistics sample programs DFH0STAT and DFH0STXR, that will report the additional TCB mode and pool. TCPIP global statistics also report whether SSL caching is taking place locally or across a sysplex.

Changes to CICS utilities

There are changes to the statistics utility program DFHSTUP to report both the additional TCB mode and TCB pool.

Changes to problem determination

The message DFHSO0123 has a new insert that is used when CICS identifies a revoked certificate during an encryption negotiation. CICS closes the connection immediately and issues the message.

\# A new message DFHSO0128 is issued when the RACF profile specified in the
\# CRLPROFILE system initialization parameter is inaccessible. This could be because
\# the CICS region userid is not authorized to access the information in the profile or
\# the named profile does not exist or is incomplete. CICS continues to initialize
\# without CRL support.

\# A new message DFHSO0129 is issued when the LDAP server that is specified in
\# the RACF profile for certificate revocation lists becomes inactive. CICS continues to
\# initialize and CRL support is disabled. CRL support can only be re-enabled by
\# recycling CICS.

Security

Security has been enhanced through support for the new AES ciphers and higher encryption keylengths. You can also restrict the minimum encryption level for negotiations between clients and CICS by editing the default values of cipher suites that are used in SSL connections.

The ENCRYPTION system initialization parameter default value is now STRONG, raising the level of security in CICS when using SSL.

The security category for the CCRL transaction is category 2.

Migration

Migration of existing functions

The default setting for the ENCRYPTION system initialization parameter has changed to STRONG. If you have no high encryption ciphers installed (security level 3) on z/OS, then you need to downgrade the default setting for the ENCRYPTION system initialization parameter. The NORMAL setting that has been used as the default in previous releases, has changed to MEDIUM for this release of CICS. For migration purposes, NORMAL is accepted as an alternative to MEDIUM.

The SSLTCBS system initialization parameter is now obsolete and has been replaced by MAXSSLTCBS. MAXSSLTCBS controls the maximum number of S8 TCBs that are allowed to run concurrently in the open transaction environment (OTE) TCB pool for SSL.

Migration to the new function

You can exploit the CIPHERS attribute to better control the encryption negotiation process between CICS and clients. You can use the CRLPROFILE and SSLCACHE system initialization parameters to verify certificates in the SSL handshake and improve the performance of the handshake through sharing the SSL cache across CICS regions.

Coexistence

CICS continues to support SSL v3.0 as well as the new TLS 1.0 protocol.

CICSPlex SM support

There are a number of changes to CICSPlex SM interfaces and API to match the new support for SSL enhancements.

Changes to CICSPlex SM end user interface views

There are no new end user interface views in CICS Transaction Server for z/OS, Version 3 Release 1. However, changes have been made to match the new support for SSL, to the following views:
- "EJCODEF view"
- "TCPDEF view" on page 190
- "EJCOSE view" on page 190
- "TCPIPS view" on page 190

EJCODEF view

The following attribute has been added to the EJCODEF view:

CIPHERS
    SSL cipher suite codes

The **OUTPRIVACY** attribute in this view is no longer valid in CICS Transaction Server 3.1 and, although it can be seen, it will be ignored.

TCPDEF view

The following attribute has been added to the TCPDEF view:

CIPHERS
SSL cipher suite codes

The **PRIVACY** attribute in this view is no longer valid in CICS Transaction Server 3.1 and, although it can be seen, it will be ignored.

EJCOSE view

The following attributes have been added to the EJCOSE view:

CIPHERS
SSL cipher suite codes

NUMCIPHERS
The number of cipher suites that are used to negotiate encryption levels.

TCPIPS view

The following attributes have been added to the TCPIPS view:

CIPHERS
SSL cipher suite codes

NUMCIPHERS
The number of cipher suites that are used to negotiate encryption levels.

Changes to the CICSPlex SM application programming interface

Changes have been made to the following resource tables:
- "EJCODEF resource table"
- "TCPDEF resource table"
- "EJCOSE resource table" on page 191
- "TCPIPS resource table" on page 191
- "TCPIPGBL resource table" on page 191
- "TASK resource table" on page 191
- "CICSRGN resource table" on page 191

EJCODEF resource table

A new attribute is added to the EJCODEF resource table:

CIPHERS
Shows the 56-characters list of SSL cipher suite codes

TCPDEF resource table

A new attribute is added to the TCPDEF resource table:

CIPHERS
Shows the 56-characters list of SSL cipher suite codes

EJCOSE resource table

New attributes are added to the EJCOSE resource table:

CIPHERS
> Shows the 56-characters list of SSL cipher suite codes

NUMCIPHERS
> The number of cipher suites that are used to negotiate encryption levels.

TCPIPS resource table

New attributes are added to the TCPIPS resource table:

CIPHERS
> Shows the 56-characters list of SSL cipher suite codes

NUMCIPHERS
> The number of cipher suites that are used to negotiate encryption levels.

TCPIPGBL resource table

New attributes are added to the TCPIPGBL resource table:

\# **CRLPROFILE**
\#
\# > Name of the RACF profile that specifies the LDAP server that contains
> certificate revocation lists (CRLs) and authorizes CICS to access them.

SSLCACHE
> SSL cache type - local or sysplex

TASK resource table

New attributes are added to the TASK resource table:

TMRSTDLY
> Shows the 8-byte time the task waited for redispatch after the Dispatcher
> change-TCB mode request was issued

TMRCMDLY
> Shows the 8-byte time the task waited to obtain a CICS SSL TCB, because
> the system had reached the limit set by MAXSSLTCBS

CICSRGN resource table

The CICSRGN resource table includes the following SPI attributes:

MAXSSLTCBS
> Shows the current maximum number of TCBs in the SSL OTE pool.

ACTSSLTCBS
> Shows the number of currently allocated SSL pool TCBs.

Changes to CICSPlex SM Web User Interface

Changes have been made to the following views:
- "CorbaServer definition view" on page 192
- "TCP/IP Service Definition view" on page 192
- "CorbaServer view" on page 192
- "TCP/IP service view" on page 192

- "TCP/IP global status view"
- "Clocks and timings view" on page 193
- "CICS region view" on page 193

CorbaServer definition view

The following attribute has been added to the EJCODEF (EYUSTARTEJCODEF) view:

CIPHERS
     SSL cipher suite codes

The **OUTPRIVACY** attribute in this view is no longer valid in CICS TS 3.1 and, although it can be viewed, it will be ignored.

TCP/IP Service Definition view

The following attribute has been added to the TCPDEF (EYUSTARTTCPDEF) view:

CIPHERS
     SSL cipher suite codes

The **PRIVACY** attribute in this view is no longer valid in CICS TS 3.1 and, although it can be viewed, it will be ignored.

CorbaServer view

The following attributes have been added to the EJCOSE (EYUSTARTEJCOSE.DETAILED) view:

CIPHERS
     SSL cipher suite codes

NUMCIPHERS
     The number of cipher suites that are used to negotiate encryption levels

TCP/IP service view

The following attributes have been added to the TCPIPS (EYUSTARTTCPIPS.DETAILED) view:

CIPHERS
     SSL cipher suite codes

NUMCIPHERS
     The number of cipher suites that are used to negotiate encryption levels

TCP/IP global status view

The following attributes have been added to the TCPIPGBL (EYUSTARTTCPIPGBL.DETAILED) view:

\#   **CRLPROFILE**
\#   Name of the RACF profile that specifies the LDAP server that contains
\#   certificate revocation lists (CRLs) and authorizes CICS to access them.

SSLCACHE
     Whether SSL is to use local or sysplex-wide caching of session ids

Clocks and timings view

The following attributes have been added to the Clocks and timings view, one of the Active task view set (EYUSTARTTASK.DETAIL2) within the Task operations view:

TMRCMDLY
> CICS TCB Change Mode delay time

TMRSTDLY
> Maximum CICS SSL TCB delay time

CICS region view

The following statistics attributes have been added to the CICSRGN (EYUSTARTCICSRGN.DETAILED) view:

MAXSSLTCBS
> Shows the current maximum number of TCBs in the SSL OTE pool.

ACTSSLTCBS
> Shows the number of currently allocated SSL pool TCBs.

Part 3. Application transformation

CICS Transaction Server for z/OS, Version 3 Release 1 provides a range of new functions that enable you to enhance your existing applications, and to construct new applications, using contemporary programming languages, constructs, and tools.

Chapter 7. Enhanced inter-program data transfer: channels as modern-day COMMAREAs

Traditionally, CICS programs have used communication areas (COMMAREAs) to exchange data. In CICS Transaction Server for z/OS, Version 3 Release 1, *channels* and *containers* provide an improved method of transferring data between programs, in amounts that far exceed the 32KB limit that applies to COMMAREAs.

- A *container* is a named block of data designed for passing information between programs.
- Containers are grouped together in sets called *channels.* A channel is a standard mechanism for exchanging data between CICS programs, and is analogous to a parameter list. A channel can be used on the LINK, START, XCTL, and RETURN commands, and with local and remote transactions. There is no limit to the number of containers that can be added to a channel, and the size of each container is limited only by the amount of storage available.

Benefits of channels

The channel/container model has several advantages over the communication areas (COMMAREAs) traditionally used by CICS programs to exchange data. For example:

- Unlike COMMAREAs, channels are not limited in size to 32KB. There is no limit to the number of containers that can be added to a channel, and the size of individual containers is limited only by the amount of storage that you have available.

  # Take care not to create so many large containers that you limit the amount of
  # storage available to other applications.

- Because a channel can comprise multiple containers, it can be used to pass data in a more structured way. In contrast, a COMMAREA is a monolithic block of data.
- Unlike COMMAREAs, channels don't require the programs that use them to know the exact size of the data returned.
- A channel is a standard mechanism for exchanging data between CICS programs. A channel can be passed on LINK, START, XCTL, and RETURN commands. Distributed program link (DPL) is supported, and the transactions started by START CHANNEL and RETURN TRANSID commands may be remote.
- Channels can be used by CICS application programs written in any of the CICS-supported languages. For example, a Java client program on one CICS region can use a channel to exchange data with a COBOL server program on a back-end AOR.
- A server program can be written to handle multiple channels. It can, for example:
  1. Discover, dynamically, the channel that it was invoked with
  2. Browse the containers in the channel
  3. Vary its processing according to the channel it's been passed
- You can build "components" from sets of related programs invoked through one or more channels.
- The loose coupling between clients and components permits easy evolution. Clients and components can be upgraded at different times. For example, first a component could be upgraded to handle a new channel, then the client program upgraded (or a new client written) to use the new channel.

- The programmer is relieved of storage management concerns. CICS automatically destroys containers (and their storage) when they go out of scope.
- The data conversion model used by channel applications is much simpler than that used by COMMAREA applications. Also, whereas in COMMAREA applications data conversion is controlled by the *system* programmer, in channel applications it is controlled by the *application* programmer, using simple API commands.
- Programmers with experience of CICS business transaction services (BTS) will find it easy to use containers in non-BTS applications.
- Programs that use containers can be called from both channel and BTS applications.
- Non-BTS applications that use containers can be migrated into full BTS applications. (They form a migration route to BTS.)

This topic has listed some of the many benefits of channels. However, channels may not be the best solution in all cases. When designing an application, there are one or two implications of using channels that you should be aware of:

- When a channel is to be passed to a remote program or transaction, passing a large amount of data may affect performance. This is particularly true if the local and remote regions are connected by an ISC, rather than MRO, connection.
- A channel may use more storage than a COMMAREA designed to pass the same data. This is because:
  1. Container data can be held in more than one place.
  2. COMMAREAs are accessed by pointer, whereas the data in containers is copied between programs.

Terminology

channel
> A group of **containers**, used to pass data between programs. A channel is analogous to a parameter list.

container
> A named block of data used to pass information between programs. You can think of it as a "named communication area (COMMAREA)". Programs can pass any number of containers between each other. Containers are grouped together in sets called **channels**.

current channel
> The channel, if any, with which a program is invoked. The current channel, for a particular invocation of a particular program, does not change.

Channels: quick start

Containers and channels

Containers are named blocks of data designed for passing information between programs. You can think of them as "named communication areas (COMMAREAs)". Programs can pass any number of containers between each other and the size of the containers is limited only by the amount of storage available.

Containers are grouped together in sets called **channels**. A channel is analogous to a parameter list.

To create named containers and assign them to a channel, a program uses EXEC CICS PUT CONTAINER(*container-name*) CHANNEL(*channel-name*) commands. It can then pass the channel (and its containers) to a second program using the CHANNEL(*channel-name*) option of the EXEC CICS LINK, XCTL, START, or RETURN commands.

The second program can read containers passed to it using the EXEC CICS GET CONTAINER(*container-name*) command. This command reads the named container belonging to the channel that the program was invoked with.

If the second program is invoked by a LINK command, it can also return containers to the calling program. It can do this by creating new containers, or by reusing existing containers.

Channels and containers are visible only to the program that creates them and the programs they are passed to. When these programs terminate, CICS automatically destroys the containers and their storage.

Channel containers are not recoverable. If you need to use recoverable containers, use CICS business transaction services (BTS) containers. The relationship between channel and BTS containers is described in "Channels and BTS activities" on page 216.

Basic examples

Figure 17 on page 201 shows a COBOL program, `CLIENT1`, that:

1. Uses PUT CONTAINER(*container-name*) CHANNEL(*channel-name*) commands to create a channel called `inqcustrec` and add two containers, `custno` and `branchno`, to it; these contain a customer number and a branch number, respectively.
2. Uses a LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) command to link to program `SERVER1`, passing the `inqcustrec` channel.
3. Uses a GET CONTAINER(*container-name*) CHANNEL(*channel-name*) command to retrieve the customer record returned by `SERVER1`. The customer record is in the `custrec` container of the `inqcustrec` channel.

Note that the same COBOL copybook, `INQINTC`, is used by both the client and server programs. Line 3 and lines 5 through 7 of the copybook represent the `INQUIRY-CHANNEL` and its containers. These lines are not strictly necessary to the working of the programs, because containers and channels are created simply by being *named* (on, for example, PUT CONTAINER commands); they do not have to be defined. However, the inclusion of these lines in the copybook used by both programs makes for easier maintenance; they record the names of the containers used.

```
#                     *              copybook INQINTC
#                     * Channel name
#                       01 INQUIRY-CHANNEL  PIC X(16) VALUE 'inqcustrec'.
#                     * Container names
#                       01 CUSTOMER-NO        PIC X(16) VALUE 'custno'.
#                       01 BRANCH-NO          PIC X(16) VALUE 'branchno'.
#                       01 CUSTOMER-RECORD    PIC X(16) VALUE 'custrec'.
#                     * Define the data fields used by the program
#                       01 CUSTNO     PIC X(8).
#                       01 BRANCHNO   PIC X(5).
#                       01 CREC.
#                         02 CUSTNAME   PIC X(80).
#                         02 CUSTADDR1  PIC X(80).
#                         02 CUSTADDR2  PIC X(80).
#                         02 CUSTADDR3  PIC X(80).
#
#                     Figure 16. Copybook INQINTC
#
```

Figure 16. Copybook INQINTC

Recommendation

For ease of maintenance of a client/server application that uses a channel, create a copybook that records the names of the containers used and defines the data fields that map to the containers. Include the copybook in both the client and the server program.

Note: This example shows two COBOL programs. The same techniques can be used in any of the other languages supported by CICS. However, *for COBOL programs only*, if the server program uses the SET option (instead of INTO) on the **EXEC CICS GET CONTAINER** command, the structure of the storage pointed to by SET must be defined in the LINKAGE section of the program. This means that you will require two copybooks rather than one. The first, in the WORKING-STORAGE section of the program, names the channel and containers used. The second, in the LINKAGE section, defines the storage structure.

```
               IDENTIFICATION DIVISION.
               PROGRAM-ID. CLIENT1.

               WORKING-STORAGE SECTION.

                 COPY INQINTC

               PROCEDURE DIVISION.
               MAIN-PROCESSING SECTION.

               *
               * INITIALISE CUSTOMER RECORD
               *
                   ... CREATE CUSTNO and BRANCHNO
               *
               * GET CUSTOMER RECORD
               *
                   EXEC CICS PUT CONTAINER(CUSTOMER-NO) CHANNEL(INQUIRY-CHANNEL)
                             FROM(CUSTNO) FLENGTH(LENGTH OF CUSTNO)
                             END-EXEC
                   EXEC CICS PUT CONTAINER(BRANCH-NO) CHANNEL(INQUIRY-CHANNEL)
                             FROM(BRANCHNO) FLENGTH(LENGTH OF BRANCHNO)
                             END-EXEC

                   EXEC CICS LINK PROGRAM('SERVER1') CHANNEL(INQUIRY-CHANNEL) END-EXEC

                   EXEC CICS GET CONTAINER(CUSTOMER-RECORD) CHANNEL(INQUIRY-CHANNEL)
                             INTO(CREC) END-EXEC

               *
               * PROCESS CUSTOMER RECORD
               *
                   ... FURTHER PROCESSING USING CUSTNAME and CUSTADDR1 etc...

                   EXEC CICS RETURN END-EXEC

                   EXIT.
```

Figure 17. A simple example of a program that creates a channel and passes it to a second program

Figure 18 on page 202 shows the SERVER1 program linked to by CLIENT1. SERVER1 retrieves the data from the custno and branchno containers it has been passed, and uses it to locate the full customer record in its database. It then creates a new container, custrec, on the same channel, and returns the customer record in it.

Note that the programmer hasn't specified the CHANNEL keyword on the GET and PUT commands in SERVER1: if the channel isn't specified explicitly, the current channel is used—that is, the channel that the program was invoked with.

```
                    IDENTIFICATION DIVISION.
                    PROGRAM-ID. SERVER1.

                    WORKING-STORAGE SECTION.

                      COPY INQINTC

                    PROCEDURE DIVISION.
                    MAIN-PROCESSING SECTION.

                        EXEC CICS GET CONTAINER(CUSTOMER-NO)
                                         INTO(CUSTNO) END-EXEC
                        EXEC CICS GET CONTAINER(BRANCH-NO)
                                         INTO(BRANCHNO) END-EXEC

                          ... USE CUSTNO AND BRANCHNO TO FIND CREC IN A DATABASE

                        EXEC CICS PUT CONTAINER(CUSTOMER-RECORD)
                                    FROM(CREC)
                                    FLENGTH(LENGTH OF CREC) END-EXEC

                        EXEC CICS RETURN END-EXEC

                        EXIT.
```

Figure 18. A simple example of a linked to program that retrieves data from the channel it has been passed. This program is linked-to by program CLIENT1 shown in Figure 17 on page 201.

Using channels: some typical scenarios

Channels and containers provide a powerful way to pass data between programs. This section contains some examples of how channels can be used.

One channel, one program

Figure 19 shows the simplest scenario—a standalone program with a single channel with which the program can be invoked.


```
EXEC CICS LINK PROGRAM('payr')
          CHANNEL('payroll')
```
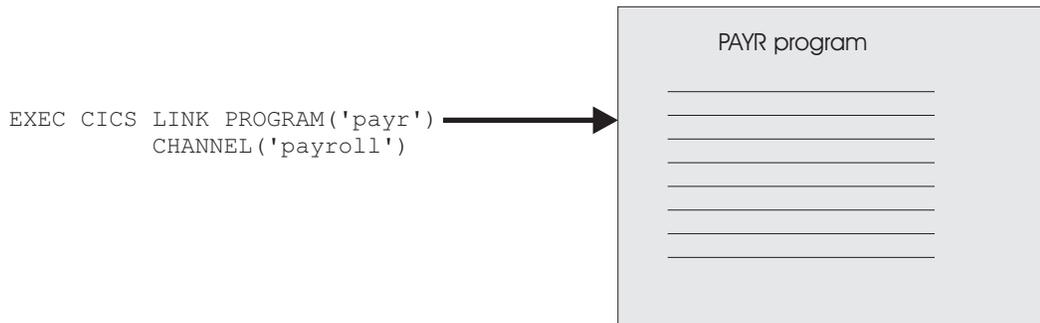
Figure 19. A standalone program with a single channel

One channel, several programs (a component)

In Figure 20 on page 203, there is a single channel to the top-level program in a set of inter-related programs. The set of programs within the shaded area can be regarded as a *component*. The client program "sees" only the external channel and has no knowledge of the processing that takes place nor of the existence of the back-end programs.

Inside the component, the programs can pass the channel between themselves. Alternatively, a component program could, for example, pass a subset of the original channel, by creating a new channel and adding one or more containers from the original channel.


```
EXEC CICS LINK PROGRAM('payr')
        CHANNEL('payroll')
```
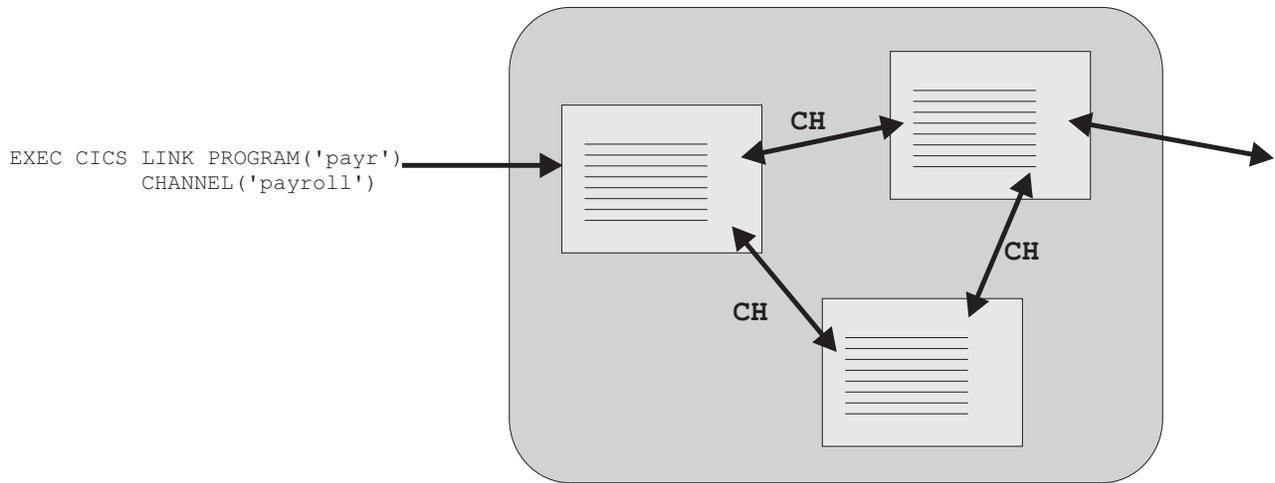
Figure 20. A "component"—a set of related programs invoked through a single external channel. "CH" indicates that the programs within the component can pass channels between themselves.

Several channels, one component

As in Figure 20, we have a set of inter-related programs that can be regarded as a component. However, this time there are two, alternative, external channels with which the component can be invoked. The top-level program in the component issues an **EXEC CICS ASSIGN CHANNEL** command to determine which channel it has been invoked with, and tailors its processing accordingly.

The "loose coupling" between the client program and the component permits easy evolution. That is, the client and the component can be upgraded at different times. For example, first the component could be upgraded to handle a third channel, consisting of a different set of containers from the first or second channels. Next, the client program could be upgraded (or a new client written) to pass the third channel.


```
EXEC CICS LINK PROGRAM('payr')
    CHANNEL('payroll-2003')
```

```
EXEC CICS LINK PROGRAM('payr')
    CHANNEL('payroll-2004')
```

```
EXEC CICS ASSIGN
        CHANNEL(ch-name)
```
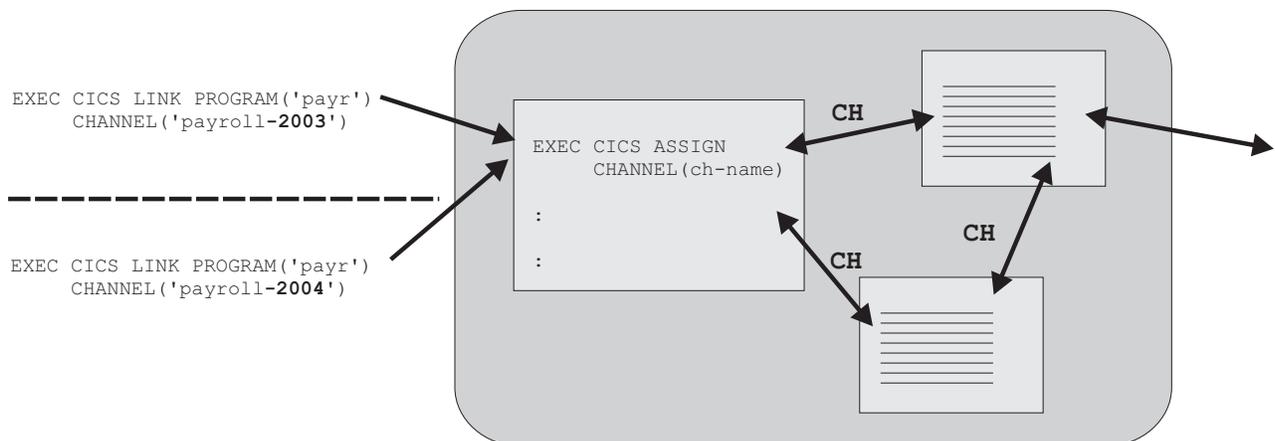
Figure 21. Multiple external channels to the same component. "CH" indicates that the programs within the component may pass channels between themselves.

Multiple interactive components

Figure 22 shows a "Human resources" component and a "Payroll" component, each with a channel with which it can be invoked. The Payroll component is invoked from both a standalone program and the Human resources component.
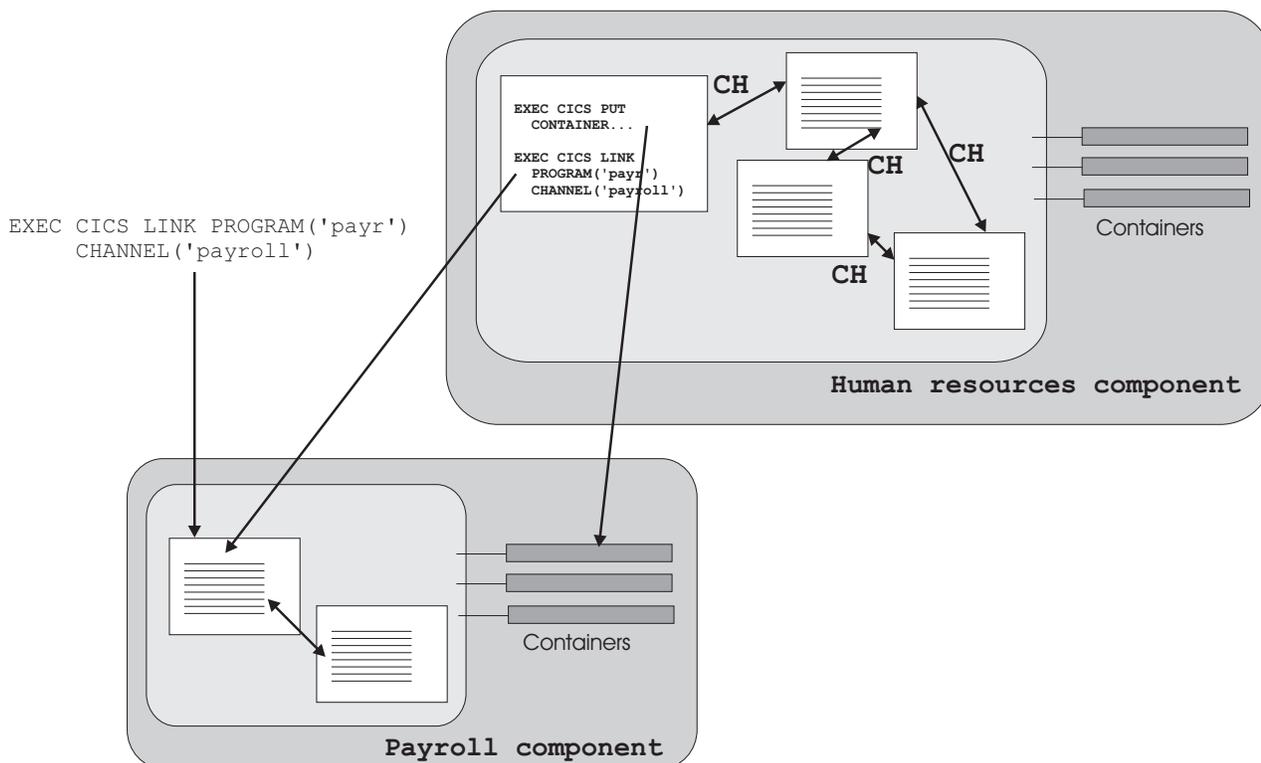
Figure 22. Multiple components which interact through their channels

Creating a channel

You create a channel by naming it on one of the following commands:

```
EXEC CICS LINK PROGRAM CHANNEL
EXEC CICS MOVE CONTAINER CHANNEL TOCHANNEL
EXEC CICS PUT CONTAINER CHANNEL
EXEC CICS RETURN TRANSID CHANNEL
EXEC CICS START TRANSID CHANNEL
EXEC CICS XCTL PROGRAM CHANNEL
```

If the channel doesn't already exist, within the current program scope, it is created.

The most straightforward way to create a channel, and populate it with containers of data, is to issue a succession of **EXEC CICS PUT CONTAINER**(*container-name*) CHANNEL(*channel-name*) FROM(*data_area*) commands. The first PUT command creates the channel (if it doesn't already exist), and adds a container to it; the subsequent commands add further containers to the channel. If the containers already exist, their contents are overwritten by the new data.

An alternative way to add containers to a channel is to move them from another channel. To do this, use the following command:

```
EXEC CICS MOVE CONTAINER(container-name) AS(container-new-name)
     CHANNEL(channel-name1) TOCHANNEL(channel-name2)
```

Note:

1. If the `CHANNEL` or `TOCHANNEL` option isn't specified, the current channel is implied—see "The current channel."

2. The source channel must be in program scope.

3. If the target channel doesn't already exist, within the current program scope, it is created.

4. If the source container doesn't exist, an error occurs.

5. If the target container doesn't already exist, it is created; if it already exists, its contents are overwritten.

You can use MOVE CONTAINER, instead of GET CONTAINER and PUT CONTAINER, as a more efficient way of transferring data between channels.

If the channel named on the following commands doesn't already exist, within the current program scope, an *empty* channel is created:

- **EXEC CICS LINK** PROGRAM CHANNEL(*channel-name*)
- **EXEC CICS RETURN** TRANSID CHANNEL(*channel-name*)
- **EXEC CICS START** TRANSID CHANNEL(*channel-name*)
- **EXEC CICS XCTL** PROGRAM CHANNEL(*channel-name*)

The current channel

A program's *current channel* is the channel (if any) with which it was invoked. The program can create other channels. However, the current channel, for a particular invocation of a particular program, does not change. It is analogous to a parameter list.

Current channel example, with LINK commands

Figure 23 on page 206 illustrates the origin of a program's current channel. It shows five interactive programs. Program A is a top-level program started by, for example, a terminal end user. It isn't started by a program and doesn't have a current channel.

B, C, D, and E are second-, third-, fourth-, and fifth-level programs, respectively.

Program B's current channel is X, passed by the CHANNEL option on the **EXEC CICS LINK** command issued by program A. Program B modifies channel X by adding one container and deleting another.

Program C's current channel is also X, passed by the CHANNEL option on the **EXEC CICS LINK** command issued by program B.

Program D has no current channel, because C doesn't pass it one.

Program E's current channel is Y, passed by the CHANNEL option on the **EXEC CICS LINK** command issued by D.

```
                    Current CH: none

                         PROG A

PUT CONTAINER(ONE) CHANNEL(X) FROM(area1)
PUT CONTAINER(TWO) CHANNEL(X) FROM(area2)
LINK PROGRAM(B) CHANNEL(X)
:
:
```

```
                    Current CH: X

                         PROG B

:
PUT CONTAINER(THREE) FROM(area-a)
DELETE CONTAINER(TWO)
LINK PROGRAM(C) CHANNEL(X)
:
:
RETURN
```

```
                    Current CH: X
                         PROG C
:
LINK PROGRAM(D)
:
:
RETURN
```

```
                    Current CH: none
                         PROG D
:
PUT CONTAINER(first) CHANNEL(Y) FROM(a1)
PUT CONTAINER(second) CHANNEL(Y) FROM(a2)
LINK PROGRAM(E) CHANNEL(Y)
:
RETURN
```

```
                    Current CH: Y
                         PROG E

:
:
RETURN
```

Figure 23. Current channels—example with LINK commands

Table 1 on page 207 lists the name of the current channel (if any) of each of the
five programs shown in Figure 23.

Table 1. The current channels of interactive programs—example with LINK commands

Prog.	Current CH	Issues commands	Comments
A	None	```.
EXEC CICS PUT CONTAINER(ONE)
CHANNEL(X) FROM(area1)
EXEC CICS PUT CONTAINER(TWO)
CHANNEL(X) FROM(area2)
EXEC CICS LINK PROGRAM(B) CHANNEL(X)
.```	Program A creates channel X and passes it to program B. Note that, by the time control is returned to program A by program B, the X channel has been modified—it doesn't contain the same set of containers as when it was created by program A. (Container TWO has been deleted and container THREE added by program B.)
B	X	```.
EXEC CICS PUT CONTAINER(THREE)
FROM(area-a)
EXEC CICS DELETE CONTAINER(TWO)
EXEC CICS LINK PROGRAM(C) CHANNEL(X)
.
.
EXEC CICS RETURN```	Program B modifies channel X (its current channel) by adding and deleting containers, and passes the modified channel to program C. Program B doesn't need to specify the CHANNEL option on the PUT CONTAINER and DELETE CONTAINER commands; its current channel is implied.
C	X	```.
EXEC CICS LINK PROGRAM(D)
.
.
EXEC CICS RETURN```	Program C links to program D, but does not pass it a channel.
D	None	```.
EXEC CICS PUT CONTAINER(first)
CHANNEL(Y) FROM(a1)
EXEC CICS PUT CONTAINER(second)
CHANNEL(Y) FROM(a2)
EXEC CICS LINK PROGRAM(E) CHANNEL(Y)
.
.
EXEC CICS RETURN```	Program D creates a new channel, Y, which it passes to program E.
E	Y	```.
RETURN
.```	Program E performs some processing on the data it's been passed and returns.

Current channel example, with XCTL commands

Figure 24 on page 208 shows four interactive programs. A1 is a top-level program started by, for example, a terminal end user. It isn't started by a program and doesn't have a current channel. B1, B2, and B3 are all second-level programs.

B1's current channel is X, passed by the CHANNEL option on the **EXEC CICS LINK** command issued by A1.

B2 has no current channel, because B1 doesn't pass it one.

B3's current channel is Y, passed by the CHANNEL option on the **EXEC CICS XCTL** command issued by B2.

When B3 returns, channel Y and its containers are deleted by CICS.
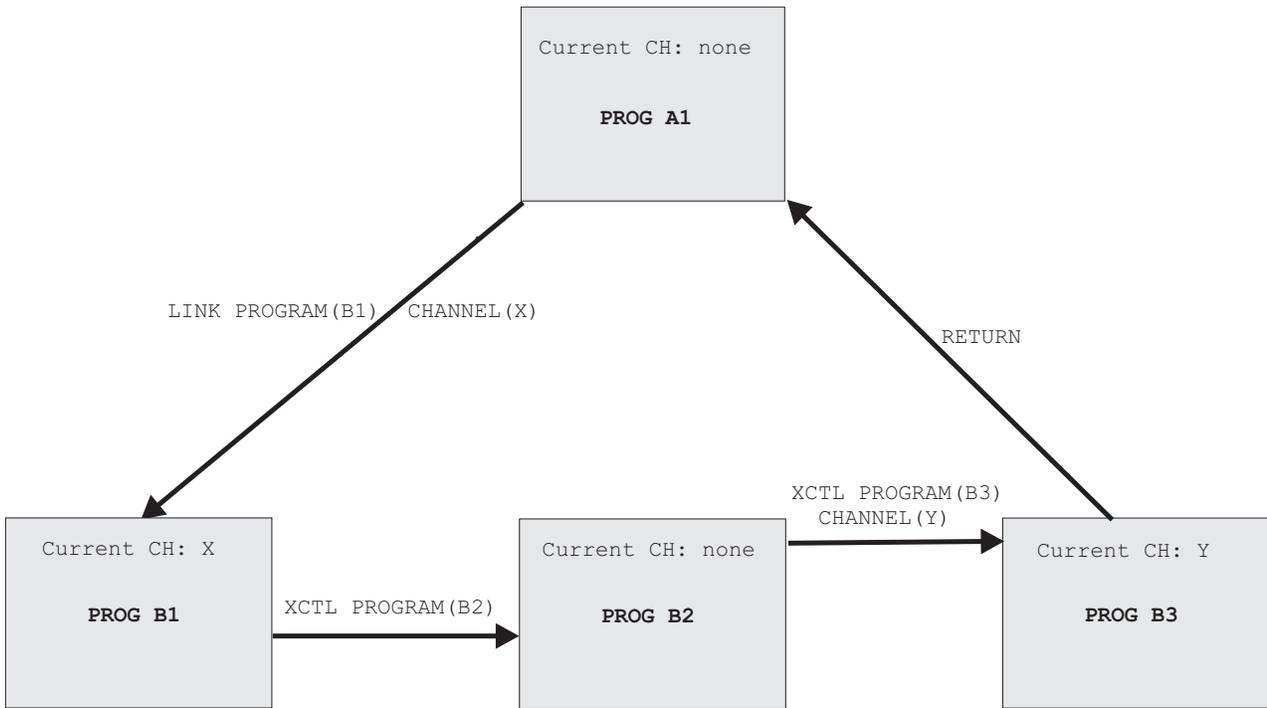
Figure 24. Current channels—example, with XCTL commands

Table 2 lists the name of the current channel (if any) of each of the four programs shown in Figure 24.

Table 2. The current channels of interactive programs—example

Program	Current channel	Issues command
A1	None	. EXEC CICS LINK PROGRAM(B1) CHANNEL(X) .
B1	X	. EXEC CICS XCTL PROGRAM(B2) .
B2	None	. EXEC CICS XCTL PROGRAM(B3) CHANNEL(Y) .
B3	Y	. EXEC CICS RETURN .

Current channel: START and RETURN commands

Besides **EXEC CICS LINK** and **XCTL**, two other commands can be used to invoke a program and pass it a channel:

EXEC CICS START TRANSID(*tranid***) CHANNEL(***channel-name***)**
> The program that implements the started transaction (or the first program, if there are more than one) is passed the channel, which becomes its current channel.

```
EXEC CICS RETURN TRANSID(tranid) CHANNEL(channel-name)
```
The CHANNEL option is valid only:
- On pseudoconversational RETURNs—that is, on RETURN commands that specify, on the TRANSID option, the next transaction to be run at the user terminal.
- If issued by a program at the highest logical level—that is, a program that returns control to CICS.

The program that implements the next transaction is passed the channel, which becomes its current channel.

The scope of a channel

The *scope* of a channel is the code from which it can be accessed.

Scope example, with LINK commands

Figure 25 on page 210 shows the same five interactive programs described in "Current channel example, with LINK commands" on page 205.

The scope of the X channel—the code from which it can be accessed—is programs A, B, and C.

The scope of the Y channel is programs D and E.

Note that, by the time control is returned to program A by program B, the X channel has been modified—it doesn't contain the same set of containers as when it was created by program A.

```
                    Current Channel: none

                           PROG A
Scope of Channel X
PUT CONTAINER(ONE) CHANNEL(X) FROM(area1)
PUT CONTAINER(TWO) CHANNEL(X) FROM(area2)
LINK PROGRAM(B) CHANNEL(X)
:
:
```

```
                    Current Channel: X

                           PROG B

:
PUT CONTAINER(THREE) FROM(area-a)
DELETE CONTAINER(TWO)
LINK PROGRAM(C) CHANNEL(X)
:
:
RETURN
```

```
                    Current Channel: X

                           PROG C
:
LINK PROGRAM(D)
:
:
RETURN
```

```
                    Current Channel: none
                           PROG D
Scope of Channel Y   :
PUT CONTAINER(first) CHANNEL(Y) FROM(a1)
PUT CONTAINER(second) CHANNEL(Y) FROM(a2)
LINK PROGRAM(E) CHANNEL(Y)
:
RETURN
```

```
                    Current Channel: Y

                           PROG E

:
:
RETURN
```
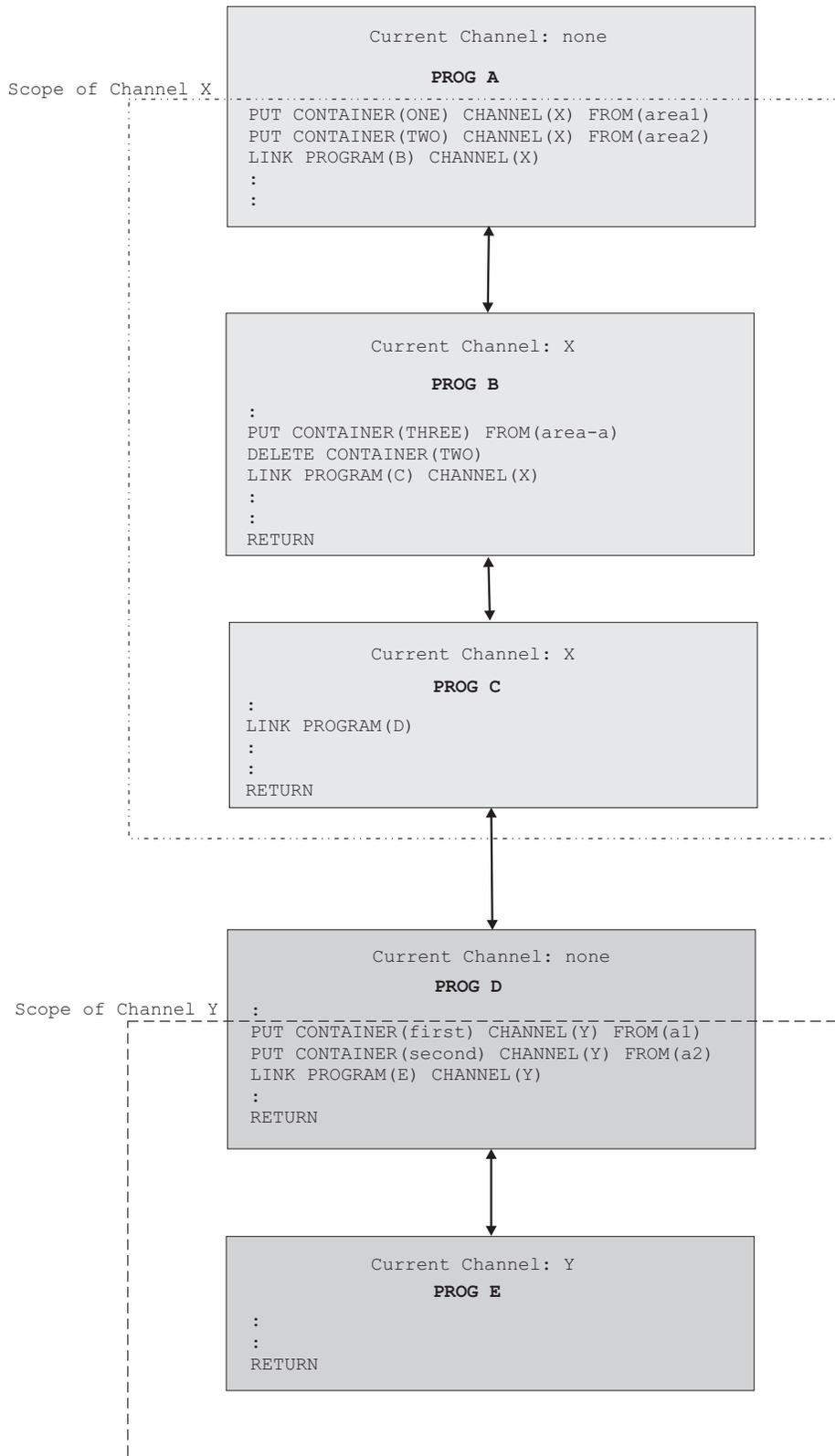
Figure 25. The scope of a channel—example showing LINK commands

Table 3 on page 211 lists the name and scope of the current channel (if any) of each of the five programs shown in Figure 25.

Table 3. The scope of a channel—example with LINK commands

Program	Current channel	Scope of channel
A	None	Not applicable
B	X	A, B, C
C	X	A, B, C
D	None	Not applicable
E	Y	D, E

Scope example, with LINK and XCTL commands

Figure 26 on page 212 shows the same four interactive programs described in "Current channel example, with XCTL commands" on page 207, plus a third-level program, C1, that is invoked by an **EXEC CICS LINK** command from program B1.

The scope of the X channel is restricted to A1 and B1.

The scope of the Y channel is B2 and B3.

The scope of the Z channel is B1 and C1.

Note that, by the time control is returned to program A1 by program B3, it's possible that the X channel may have been modified by program B1—it might not contain the same set of containers as when it was created by A1.

```
Channel X scope =      ▢

Channel Y scope =      ▢

Channel Z scope =      ▢

X and Z scope =        ⊞
```

Current CH: none

PROG A1

LINK PROGRAM(B1) CHANNEL(X)

RETURN

XCTL PROGRAM(B3)
CHANNEL(Y)

Current CH: X

PROG B1

XCTL PROGRAM(B2)

Current CH: none

PROG B2

Current CH: Y

PROG B3

LINK PROGRAM(C1) CHANNEL(Z)

Current CH: Z

PROG C1
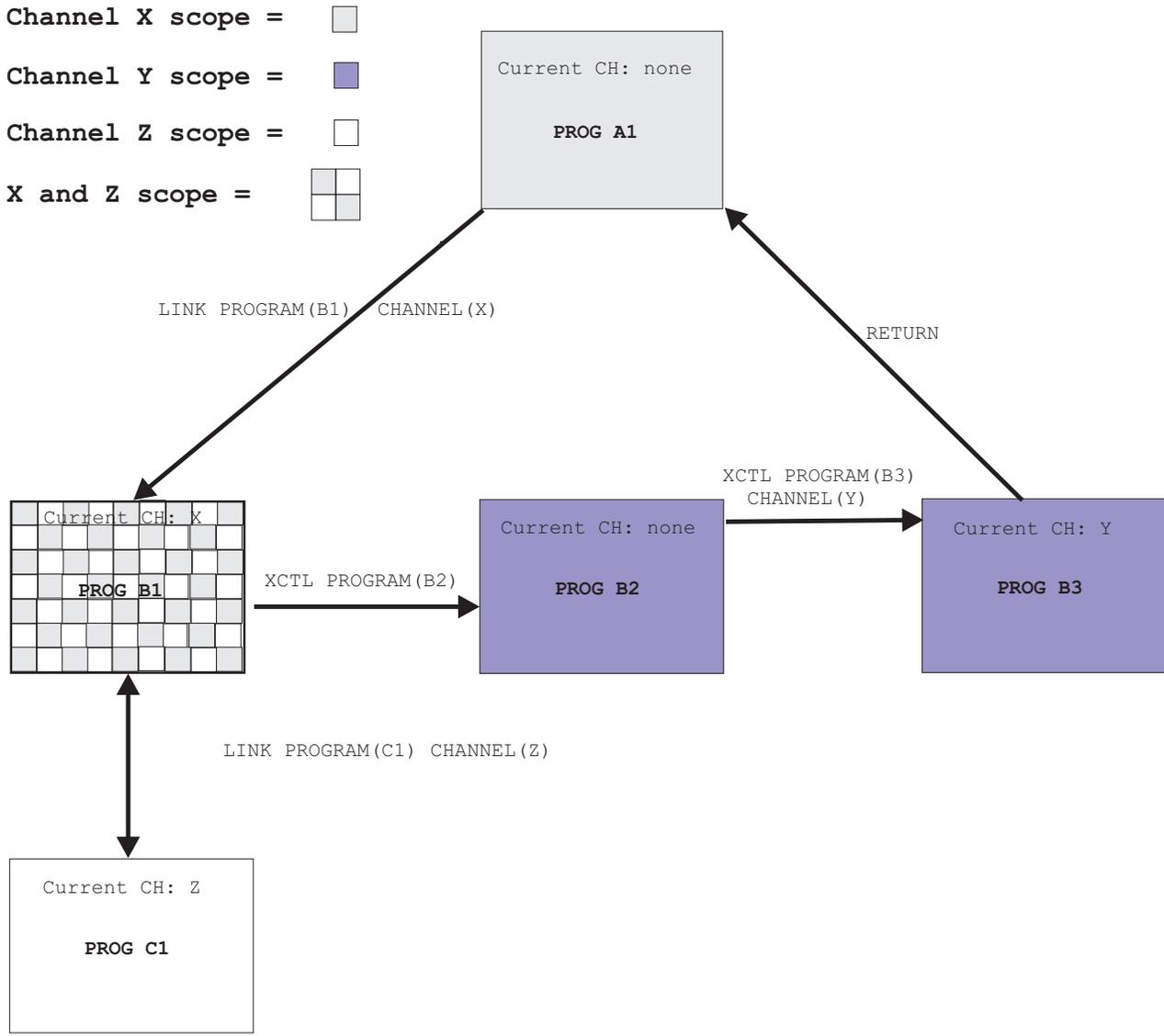
Figure 26. The scope of a channel—example showing LINK and XCTL commands

Table 4 lists the name and scope of the current channel (if any) of each of the five programs shown in Figure 26.

Table 4. The scope of a channel—example with LINK and XCTL commands

Program	Current channel	Scope of channel
A1	None	Not applicable
B1	X	A1 and B1
B2	None	Not applicable
B3	Y	B2 and B3
C1	Z	B1 and C1

Discovering which containers were passed to a program

Typically, programs that exchange a channel are written to handle that channel. That is, both client and server programs know the name of the channel, and the names and number of the containers in the channel. However, if, for example, a server program or component is written to handle more than one channel, on invocation it must discover which of the possible channels it's been passed.

A program can discover its current channel—that is, the channel with which it was invoked—by issuing an EXEC CICS ASSIGN CHANNEL command. (If there is no current channel, the command returns blanks.)

The program can also (should it need to) get the names of the containers in its current channel by browsing. Typically, this is not necessary. A program written to handle several channels is often coded to be aware of the names and number of the containers in each possible channel.

To get the names of the containers in the current channel, use the browse commands:
- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) .
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

Having retrieved the name of its current channel and, if necessary, the names of the containers in the channel, a server program can adjust its processing to suit the kind of data that it's been passed.

Discovering which containers were returned from a link

A program creates a channel, which it passes to a second program by means of an EXEC CICS LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) command. The second program performs some processing on the data it's been passed, and returns the results in the same channel (its current channel).

On return, the first program knows the name of the channel which has been returned, but not necessarily the names of the containers in the channel. (Does the returned channel contain the same containers as the passed channel, or has the second program deleted some or created others?) The first program can discover the container-names by browsing. To do this, it uses the commands:
- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) CHANNEL(*channel-name*).
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

CICS read only containers

CICS can create channels and containers for its own use, and pass them to user programs. In some cases CICS marks these containers as read only, so that the user program cannot modify data which CICS needs on return from the user program.

User programs cannot create read only containers.

You cannot overwrite, move, or delete a read only container. Thus, if you specify a read only container on a PUT CONTAINER, MOVE CONTAINER, or DELETE CONTAINER command an INVREQ condition occurs.

Designing a channel: best practices

It's possible to use containers to pass data in the same way as communication areas (COMMAREAs) have traditionally been used. However, channels have several advantages over COMMAREAs (see "Benefits of channels" on page 197) and it pays to design your channels to make the most of these improvements.

At the end of a DPL call, input containers that have not been changed by the server program are not returned to the client. Input containers whose contents have been changed by the server program, and containers created by the server program, are returned. Therefore, for optimal DPL performance:
- Use separate containers for input and output data.
- The server program, not the client, should create the output containers.
- Use separate containers for read-only and read-write data.
- If a structure is optional, make it a separate container.
- Use dedicated containers for error information.

Here are some general tips on designing a channel. They include and expand on the recommendations for achieving optimal DPL performance.
- Use separate containers for input and output data. This leads to:
    - Better encapsulation of the data, making your programs easier to maintain.
    - Greater efficiency when a channel is passed on a DPL call, because smaller containers flow in each direction.
- The server program, not the client, should create the output containers. If the client creates them, empty containers will be sent to the server region.
- Use separate containers for read-only and read-write data. This leads to:
    - A simplification of your copybook structure, making your programs easier to understand.
    - Avoidance of the problems with REORDER overlays.
    - Greater transmission efficiency between CICS regions, because read-only containers sent to a server region will not be returned.
- Use separate containers for each structure. This leads to:
    - Better encapsulation of the data, making your programs easier to understand and maintain.
    - Greater ease in changing one of the structures, because you don't need to recompile the entire component.
    - The ability to pass a subset of the channel to sub-components, by using the MOVE CONTAINER command to move containers between channels.
- If a structure is optional, make it a separate container. This leads to greater efficiency, because the structure is passed only if the container is present.
- Use dedicated containers for error information. This leads to:
    - Better documentation of what is error information.
    - Greater efficiency, because:
        1. The structure containing the error information is passed back only if an error occurs.
        2. It is more efficient to check for the presence of an error container by issuing a GET CONTAINER(*known-error-container-name*) command (and possibly receiving a NOTFOUND condition) than it is to initiate a browse of the containers in the channel.
- When you need to pass data of different types—for example, character data in `codepage1` and character data in `codepage2`—use separate containers for each

type, rather than one container with a complicated structure. This improves your ability to move between different code pages.

- When you need to pass a large amount of data, split it between multiple containers, rather than put it all into one container.

  When a channel is passed to a remote program or transaction, passing a large amount of data may affect performance. This is particularly true if the local and remote regions are connected by an ISC, rather than MRO, connection.

  CAUTION:
  Take care not to create so many large containers that you limit the amount of storage available to other applications.

For information about migrating programs that use COMMAREAs to use channels instead, see "Migrating from COMMAREAs to channels" on page 248.

Constructing and using a channel: an example

Figure 27 shows a CICS client program that:

1. Uses **EXEC CICS PUT CONTAINER** commands to construct (and put data in) a set of containers. The containers are all part of the same named channel—"payroll-2004".

2. Issues an **EXEC CICS LINK** command to invoke the PAYR server program, passing it the payroll-2004 channel.

3. Issues an **EXEC CICS GET CONTAINER** command to retrieve the server's program output, which it knows will be in the status container of the payroll-2004 channel.

```
* create the employee container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('employee') CHANNEL('payroll-2004') FROM('John Doe')

* create the wage container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('wage') CHANNEL('payroll-2004') FROM('100')

* invoke the payroll service, passing the payroll-2004 channel
EXEC CICS LINK PROGRAM('PAYR') CHANNEL('payroll-2004')

* examine the status returned on the payroll-2004 channel
EXEC CICS GET CONTAINER('status') CHANNEL('payroll-2004') INTO(stat)
```

Figure 27. How a client program can construct a channel, pass it to a server program, and retrieve the server's output

Figure 28 on page 216 shows part of the PAYR server program invoked by the client. The server program:

1. Queries the channel with which it's been invoked.

2. Issues **EXEC CICS GET CONTAINER** commands to retrieve the input from the employee and wage containers of the payroll-2004 channel.

3. Processes the input data.

4. Issues an **EXEC CICS PUT CONTAINER** command to return its output in the status container of the payroll-2004 channel.

```
              "PAYR", CICS COBOL server program

              * discover which channel I've been invoked with
              EXEC CICS ASSIGN CHANNEL(ch_name)
              :
              WHEN ch_name 'payroll-2004'
                  * my current channel is "payroll-2004"
                  * get the employee passed into this program
                  EXEC CICS GET CONTAINER('employee') INTO(emp)
                  * get the wage for this employee
                  EXEC CICS GET CONTAINER('wage') INTO(wge)
                  :
                  * process the input data
                  :
                  :
                  * return the status to the caller by creating the status container
                  * on the payroll channel and putting a value in it
                  EXEC CICS PUT CONTAINER('status') FROM('OK')
                  :
                  :
              WHEN ch_name 'payroll-2005'
                  * my current channel is "payroll-2005"
                  :
                  :
                  :
```

Figure 28. How a server program can query the channel it's been passed, retrieve data from the channel's containers, and return output to the caller

Channels and BTS activities

The PUT, GET, MOVE, and DELETE CONTAINER commands used to build and interact with a channel are similar to those used in CICS business transaction services (BTS) applications. Thus, programmers with experience of BTS will find it easy to use containers in non-BTS applications. Furthermore, server programs that use containers can be called from both channel and BTS applications. An example of this is shown in Figure 29 on page 217.

```
        CICS Channel program                        CICS BTS program

                                            DEFINE ACTIVITY('payroll')
                                              PROGRAM('payact')


 * create the employee container             * create the employee container
 * on the payroll-2004 Channel               * on the payroll-2004 Channel
 EXEC CICS PUT CONTAINER('employee')         EXEC CICS PUT CONTAINER('employee')
   CHANNEL('payroll-2004') FROM('John Doe')    ACTIVITY('payroll-2004') FROM('John Doe')

 * create the wage container                 * create the wage container
 * on the payroll-2004 Channel               * on the payroll-2004 Channel
 EXEC CICS PUT CONTAINER('wage')             EXEC CICS PUT CONTAINER('wage')
   CHANNEL('payroll-2004') FROM('100')         ACTIVITY('payroll-2004') FROM('100')

 * invoke the payroll service,               * invoke the payroll service,
 * passing the payroll-2004 Channel          * passing the payroll-2004 Channel
 EXEC CICS LINK PROGRAM('PAYR')              EXEC CICS LINK ACTIVITY('payroll-2004')
   CHANNEL('payroll-2004')

 * examine the status returned on            * examine the status returned on
 * the payroll-2004 Channel                  * the payroll-2004 Channel
 EXEC CICS GET CONTAINER('status')           EXEC CICS GET CONTAINER('status')
   CHANNEL('payroll-2004') INTO(stat)          ACTIVITY('payroll-2004') INTO(stat)


    Simple client uses a Channel to        BTS event-driven wrapper controls
    pass containers to the service         a more sophisticated application


                                                  Program PAYACT

                                            ┌→ EXEC CICS RETRIEVE EVENT(...

                                                 WHEN('....
                                                   EXEC CICS LINK PROGRAM('payr')


                         Program PAYR
            * get the employee passed into this program
            EXEC CICS GET CONTAINER('employee')           Container-aware programs can
              INTO(emp)                                   be called from both Channel
            :                                             and BTS applications
            :
            * return the status to the caller
             EXEC CICS PUT CONTAINER('status')
               FROM('OK')
```

Figure 29. Channels and BTS activities

Context

As shown in Figure 29, a program that issues container commands can be used, without change, as part of a channel application or as part of a BTS activity.

For a program to be used in both a channel and a BTS context, the container commands that it issues must not specify any options that identify them as either channel or BTS commands. The options to be avoided on each of the container commands are:

DELETE CONTAINER
        ACQACTIVITY (BTS-specific)

ACQPROCESS (BTS-specific)
ACTIVITY (BTS-specific)
CHANNEL (channel-specific)
PROCESS (BTS-specific)

GET CONTAINER
ACQACTIVITY (BTS-specific)
ACQPROCESS (BTS-specific)
ACTIVITY (BTS-specific)
CHANNEL (channel-specific)
INTOCCSID (channel-specific)
PROCESS (BTS-specific)

MOVE CONTAINER
FROMACTIVITY (BTS-specific)
CHANNEL (channel-specific)
FROMPROCESS (BTS-specific)
TOACTIVITY (BTS-specific)
TOCHANNEL (channel-specific)
TOPROCESS (BTS-specific)

PUT CONTAINER
ACQACTIVITY (BTS-specific)
ACQPROCESS (BTS-specific)
ACTIVITY (BTS-specific)
CHANNEL (channel-specific)
DATATYPE (channel-specific)
FROMCCSID (channel-specific)
PROCESS (BTS-specific)

When a container command is executed, CICS analyzes the context (channel, BTS, or neither) in which it occurs, in order to determine how to process the command. To determine the context, CICS uses the following sequence of tests:

1. Channel: does the program have a current channel?
2. BTS: is the program part of a BTS activity?
3. None: the program has no current channel and is not part of a BTS activity. It therefore has no context in which to execute container commands. The command is rejected with an INVREQ condition and a RESP2 value of 4.

Using channels from JCICS

CICS provides the following JCICS classes that CICS Java programs can use to pass and receive channels:
* `com.ibm.cics.server.Channel`
* `com.ibm.cics.server.Container`
* `com.ibm.cics.server.ContainerIterator`

CICS also provides the following exception classes for handling errors:
* `com.ibm.cics.server.CCSIDErrorException`
* `com.ibm.cics.server.ChannelErrorException`
* `com.ibm.cics.server.ContainerErrorException`

Creating channels and containers in JCICS

To create a channel, use the `createChannel()` method of the `Task` class. For example:

```
Task t=Task.getTask();
Channel custData = t.createChannel("Customer_Data");
```

The string supplied to the `createChannel` method is the name by which the `Channel` object is known to CICS. (The name is padded with spaces to 16 characters, to conform to CICS naming conventions.)

To create a new container in the channel, use the `Channel`'s `createContainer()` method. For example:

```
Container custRec = custData.createContainer("Customer_Record");
```

The string supplied to the `createContainer()` method is the name by which the `Container` object is known to CICS. (The name is padded with spaces to 16 characters, if necessary, to conform to CICS naming conventions.) If a container of the same name already exists in this channel, a `ContainerErrorException` is thrown.

Putting data into a container

To put data into a `Container` object, use the `Container.put()` method. Data can be added to a container as a byte array or a string. For example:

```
String custNo = "00054321";
byte[] custRecIn = custNo.getBytes();
custRec.put(custRecIn);
```

Or simply:

```
custRec.put("00054321");
```

Passing a channel to another program or task

To pass a channel on a program-link or transfer program control (XCTL) call, use the `link()` and `xctl()` methods of the `Program` class, respectively:

```
programX.link(custData);
```

```
programY.xctl(custData);
```

To set the next channel on a program-return call, use the `setNextChannel()` method of the `TerminalPrincipalFacility` class:

```
terminalPF.setNextChannel(custData);
```

To pass a channel on a START request, use the `issue` method of the `StartRequest` class:

```
startrequest.issue(custData);
```

Receiving the current channel

It is not necessary for a program to receive its current channel explicitly—see "Browsing the current channel" on page 220. However, a program can get its current channel from the current task; this enables it to extract containers by name:

```
Task t = Task.getTask();
Channel custData = t.getCurrentChannel();
if (custData != null) {
    Container custRec = custData.getContainer("Customer_Record");
} else {
    System.out.println("There is no Current Channel");
}
```

Getting data from a container

Use the `Container.get()` method to read the data in a container into a byte array:

```
byte[] custInfo = custRec.get();
```

Browsing the current channel

A JCICS program that is passed a channel can access all of the `Container` objects without receiving the channel explicitly. To do this, it uses a `ContainerIterator` object. (The `ContainerIterator` class implements the `java.util.Iterator` interface.) When a `Task` object is instantiated from the current task, its `containerIterator()` method returns an `Iterator` for the current channel, or null if there is no current channel. For example:

```
Task t = Task.getTask();
ContainerIterator ci = t.containerIterator();
While (ci.hasNext()) {
    Container custData = ci.next();
    // Process the container...
}
```

A JCICS example

Figure 30 shows a Java class called `Payroll` that calls a COBOL server program named `PAYR`. The `Payroll` class uses the JCICS `com.ibm.cics.server.Channel` and `com.ibm.cics.server.Container` classes to do the same things that a non-Java client program would use EXEC CICS commands to do.

```
import com.ibm.cics.server.*;
public class Payroll {
    ...
    Task t=Task.getTask();

    // create the payroll_2004 channel
    Channel payroll_2004 = t.createChannel("payroll-2004");

    // create the employee container
    Container employee = payroll_2004.createContainer("employee");

    // put the employee name into the container
    employee.put("John Doe");

    // create the wage container
    Container wage = payroll_2004.createContainer("wage");

    // put the wage into the container
    wage.put("2000");

    // Link to the PAYROLL program, passing the payroll_2004 channel
    Program p = new Program();
    p.setName("PAYR");
    p.link(payroll_2004);

    // Get the status container which has been returned
    Container status = payroll_2004.getContainer("status");

    // Get the status information
    byte[] payrollStatus = status.get();
    ...
}
```

Figure 30. Java class that uses the JCICS com.ibm.cics.server.Channel and com.ibm.cics.server.Container classes to pass a channel to a COBOL server program

Dynamic routing with channels

EXEC CICS LINK and **EXEC CICS START** commands, which can pass either COMMAREAs or channels, can be dynamically routed. Thus the following types of channel-related request can be dynamically routed:

- Program-link (DPL) requests
- Transactions started by terminal-related START requests
- Non-terminal-related START requests

If there is a channel associated with a dynamically-routed program-link or START command, the channel's name is passed to the routing program, in the DYRCHANL field of its communication area. The DYRCHANL field applies only to the three types of request listed above. For other types of request, or if there is no channel associated with the request, it contains blanks.

Note: The routing program's communication area is mapped by the DFHDYPDS DSECT.

Note that the routing program is given the *name* of the channel, not its address, and so is unable to use the DYRCHANL field to inspect or change the contents of the containers.

When a LINK or START command passes a COMMAREA rather than a channel, the routing program can, depending on the type of request, inspect or change the COMMAREA's contents. For LINK requests and transactions started by terminal-related START requests (which are handled by the *dynamic* routing program) but not for non-terminal-related START requests (which are handled by the *distributed* routing program), the routing program is given, in the DYRACMAA field of DFHDYPDS, the *address* of the application's COMMAREA, and can inspect and change its contents.

To give the routing program the same kind of functionality with channels, an application that uses a channel can create, within the channel, a special container named DFHROUTE. If the application issues a LINK or terminal-related START request (but not a non-terminal-related START request) that is to be dynamically routed, the dynamic routing program is given, in the DYRACMAA field of DFHDYPDS, the address of the DFHROUTE container, and can inspect and change its contents.

If you are changing a program to pass a channel rather than a COMMAREA, you could use its existing COMMAREA structure to map DFHROUTE.

For introductory information about dynamic and distributed routing, see the *CICS Intercommunication Guide*. For information about writing a dynamic or distributed routing program, see the *CICS Customization Guide*.

Data conversion

Why is data conversion needed?

Here are some cases in which data conversion is necessary:

- When character data is passed between platforms that use different encoding standards—for example, EBCDIC and ASCII.
- When you want to change the encoding of some character data from one Coded Character Set Identifier (CCSID) to another.

Data conversion with channels

Applications that use channels to exchange data use a simple data conversion model. Frequently, no conversion is required and, when it is, a single programming instruction can be used to tell CICS to handle it automatically.

Note the following:

- Usually, when a (non-Java) CICS TS program calls another (non-Java) CICS TS program, no data conversion is required, because both programs use EBCDIC encoding. For example, if a CICS TS C-language program calls a CICS TS COBOL program, passing it some containers holding character data, the only reason for using data conversion would be the unusual one of wanting to change the CCSID of the data.

- The data conversion model used by channel applications is much simpler than that used by COMMAREA applications. Applications that use COMMAREAs to exchange data use the traditional data conversion model described in the *CICS Family: Communicating from CICS on System/390®* manual. Conversion is done under the control of the system programmer, using the DFHCNV conversion table, the DFHCCNV conversion program and, optionally, the DFHUCNV user-replaceable conversion program.

  In contrast, the data in channel containers is converted under the control of the application programmer, using API commands.

- The application programmer is responsible only for the conversion of user data—that is, the data in containers created by his application programs. System data is converted automatically by CICS, where necessary.

- The application programmer is concerned only with the conversion of character data. The conversion of binary data (between big-endian and little-endian) is not supported.

- Your applications can use the container API as a simple means of converting character data from one code page to another—see "Using containers to do code page conversion" on page 223.

For data conversion purposes, CICS recognizes two types of data:

CHAR Character data—that is, a text string. The data in the container is converted (if necessary) to the code page of the application that retrieves it. If the application that retrieves the data is a client on an ASCII-based system, this will be an ASCII code page. If it is a CICS Transaction Server for z/OS application, it will be an EBCDIC code page.

  All the data in a container is converted as if it were a single character string. For single-byte character set (SBCS) code pages, a structure consisting of several character fields is equivalent to a single-byte character string. However, for double-byte character set (DBCS) code pages this is not the case. If you use DBCS code pages, to ensure that data conversion works correctly you must put each character string into a separate container.

BIT All non-character data—that is, everything that is not designated as being of type CHAR. The data in the container cannot be converted. This is the default value.

The API commands used for data conversion are:

-
  ```
  EXEC CICS PUT CONTAINER [CHANNEL] [DATATYPE] [FROMCCSID]
  ```
-

```
EXEC CICS GET CONTAINER [CHANNEL] [INTOCCSID]
```

For detailed information about the PUT CONTAINER (CHANNEL) command, see
"PUT CONTAINER (CHANNEL)" on page 230. For detailed information about the
GET CONTAINER (CHANNEL) command, see "GET CONTAINER (CHANNEL)" on
page 226.

How to cause CICS to convert character data automatically

1. In the *client program*, use the DATATYPE(DFHVALUE(CHAR)) option of the
   PUT CONTAINER command to specify that a container holds character data
   and that the data is eligible for conversion. For example:

   ```
   EXEC CICS PUT CONTAINER(cont_name) CHANNEL('payroll')
                 FROM(data1) DATATYPE(DFHVALUE(CHAR))
   ```

   There is no need to specify the FROMCCSID option unless the data is not in
   the default CCSID of the client platform. (For CICS TS regions, the default
   CCSID is specified on the LOCALCCSID system initialization parameter.) The
   default CCSID is implied.

2. In the *server program*, issue a GET CONTAINER command to retrieve the data
   from the program's current channel:

   ```
   EXEC CICS GET CONTAINER(cont_name) INTO(data_area1)
   ```

   The data is returned in the default CCSID of the server platform. There is no
   need to specify the INTOCCSID option unless you want the data to be
   converted to a CCSID other than the default. If the client and server platforms
   are different, data conversion takes place automatically.

3. In the *server program*, issue a PUT CONTAINER command to return a value to
   the client:

   ```
   EXEC CICS PUT CONTAINER(status) FROM(data_area2)
                 DATATYPE(DFHVALUE(CHAR))
   ```

   The DATATYPE(DFHVALUE(CHAR)) option specifies that the container holds
   character data and that the data is eligible for conversion. There is no need to
   specify the FROMCCSID option unless the data is not in the default CCSID of
   the server platform.

4. In the *client program*, issue a GET CONTAINER command to retrieve the status
   returned by the server program:

   ```
   EXEC CICS GET CONTAINER(status) CHANNEL('payroll')
                 INTO(status_area)
   ```

   The status is returned in the default CCSID of the client platform. There is no
   need to specify the INTOCCSID option unless you want the data to be
   converted to a CCSID other than the default. If the client and server platforms
   are different, data conversion takes place automatically.

Using containers to do code page conversion

Your applications can use the container API as a simple means of converting
character data from one code page to another. The following example converts data
from `codepage1` to `codepage2`:

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(DFHVALUE(CHAR))
              FROMCCSID(codepage1) FROM(input-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(codepage2)
              SET(data-ptr) FLENGTH(data-len)
```

The following example converts data from the region's default EBCDIC code page
to a specified UTF8 code page:

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(DFHVALUE(CHAR))
          FROM(ebcdic-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(utf8_ccsid)
          SET(utf8-data-ptr) FLENGTH(utf8-data-len)
```

When using the container API in this way, bear the following in mind:

- On GET CONTAINER commands, use the SET option, rather than INTO, unless the converted length is known. (You can retrieve the length of the converted data by issuing a GET CONTAINER(*cont_name*) NODATA FLENGTH*(len)* command.)
- To avoid a storage overhead, after conversion copy the converted data and delete the container.
- To avoid shipping the channel, use a temporary channel.
- All-to-all conversion is not possible. That is, a code page conversion error occurs if a specified code page and the channel's code page are an unsupported combination.

A SOAP example

A CICS TS SOAP application:

1. Retrieves a UTF8 or UTF16 message from a socket or MQ message queue.
2. Puts the message into a container, in UTF8 format.
3. Puts EBCDIC data structures into other containers on the same channel.
4. Makes a distributed program link (DPL) call to a handler program on a back-end AOR, passing the channel.

The back-end handler program, also running on CICS TS, can use **EXEC CICS GET CONTAINER** commands to retrieve the EBCDIC data structures or the messages. It can get the messages in UTF8 or UTF16, or in its own or the region's EBCDIC code page. Similarly, it can use **EXEC CICS PUT CONTAINER** commands to place data into the containers, in UTF8, UTF16, or EBCDIC.

To retrieve one of the messages in the region's EBCDIC code page, the handler can issue the command:

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg)
```

Because the INTOCCSID option is not specified, the message data is automatically converted to the region's EBCDIC code page. (This assumes that the PUT CONTAINER command used to store the message data in the channel specified a DATATYPE of CHAR; if it specified a DATATYPE of BIT, the default, no conversion is possible.)

To return some output in the region's EBCDIC code page, the handler can issue the command:

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg)
```

Because CHAR is not specified, no data conversion will be permitted. Because the FROMCCSID option is not specified, the message data is taken to be in the region's EBCDIC code page.

To retrieve one of the messages in UTF8, the handler can issue the command:

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg) INTOCCSID(utf8)
```

The INTOCCSID option is necessary to prevent automatic conversion of the data to the region's EBCDIC code page.

To return some output in UTF8, the server program can issue the command:

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg) FROMCCSID(utf8)
```

The FROMCCSID option specifies that the message data is currently in UTF8 format. Because CHAR is not specified, no data conversion will be permitted.

Requirements

There are no special hardware or software requirements to support this function.

Related information:

Changes to CICS externals

Changes to the application programming interface

New commands

There are five new commands for working with containers and channels. For more information, see:

Changed commands

The following application programming commands have been modified.

```
EXEC CICS DELETE CONTAINER (BTS)
EXEC CICS ENDBROWSE CONTAINER
EXEC CICS GET CONTAINER (BTS)
EXEC CICS GETNEXT CONTAINER
EXEC CICS HANDLE ABEND
EXEC CICS LINK PROGRAM
EXEC CICS MOVE CONTAINER (BTS)
EXEC CICS PUT CONTAINER (BTS)
EXEC CICS RETURN
EXEC CICS STARTBROWSE CONTAINER
EXEC CICS XCTL
```

New API commands

The following new application programming commands have been introduced:

- EXEC CICS DELETE CONTAINER (CHANNEL)
- EXEC CICS GET CONTAINER (CHANNEL)
- EXEC CICS MOVE CONTAINER (CHANNEL)
- EXEC CICS PUT CONTAINER (CHANNEL)
- EXEC CICS START CHANNEL

DELETE CONTAINER (CHANNEL):

Delete a named channel container.

DELETE CONTAINER (CHANNEL)

```
►►──DELETE──CONTAINER(data-value)──────────────────────────►◄
                                    └─CHANNEL(data-value)─┘
```

Conditions: CHANNELERR, CONTAINERERR, INVREQ

This command is threadsafe.

Description

DELETE CONTAINER (CHANNEL) deletes a container from a channel and
discards any data that it contains.

The container is identified by name and by the channel for which it is a
container—the channel that "owns" it. The channel that owns the container can be
identified:
* Explicitly, by specifying the CHANNEL option.
* Implicitly, by omitting the CHANNEL option. If this is omitted, the current channel
  is implied.

Options

CHANNEL(data-value)
  specifies the name (1–16 characters) of the channel that owns the container.

CONTAINER(data-value)
  specifies the name (1–16 characters) of the container to be deleted.

GET CONTAINER (CHANNEL):

Retrieve data from a named channel container.
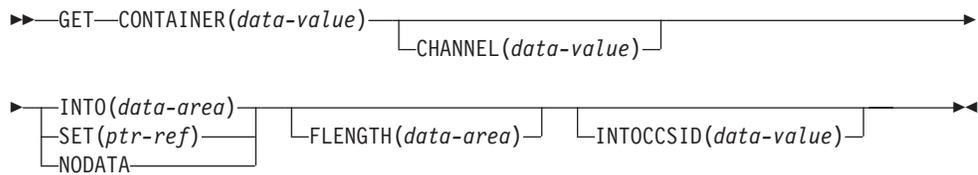
```
┌────────────────────────────────────────────────────────────────┐
│ GET CONTAINER (CHANNEL)                                         │
│                                                                │
│ ►►──GET──CONTAINER(data-value)────────────────────────────►    │
│                            └─CHANNEL(data-value)─┘             │
│                                                                │
│ ►──INTO(data-area)──────────────────────────────────────►◄    │
│   ─SET(ptr-ref)────┘ └─FLENGTH(data-area)─┘ └─INTOCCSID(data-value)─┘ │
│   ─NODATA──────────                                            │
│                                                                │
│                                                                │
│ Conditions: CCSIDERR, CHANNELERR, CONTAINERERR, INVREQ,        │
│ LENGERR                                                        │
│                                                                │
│ This command is threadsafe.                                   │
└────────────────────────────────────────────────────────────────┘
```

Description

GET CONTAINER (CHANNEL) reads the data associated with a specified channel container.

The container which holds the data is identified by name and by the channel for which it is a container—the channel that "owns" it. The channel that owns the container can be identified:

- Explicitly, by specifying the CHANNEL option.
- Implicitly, by omitting the CHANNEL option. If this is omitted, the current channel is implied.

Options

CHANNEL(data-value)
   specifies the name (1–16 characters) of the channel that owns the container.

CONTAINER(data-value)
   specifies the name (1–16 characters) of the container that holds the data to be retrieved.

FLENGTH(data-area)
   As an input field, FLENGTH specifies, as a fullword binary value, the length of the data to be read. As an output field, FLENGTH returns the length of the data in the container. Whether FLENGTH is an input or an output field depends on which of the INTO, SET, or NODATA options you specify.

   INTO option specified
           FLENGTH is both an input and an output field.

           On input, FLENGTH specifies the maximum length of the data that the program accepts. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs.

           FLENGTH need not be specified if the length can be generated by the compiler from the INTO variable. If you specify both INTO and FLENGTH, FLENGTH specifies the maximum length of the data that the program accepts.

On **output** (that is, on completion of the retrieval operation) CICS sets the data area, if specified, to the actual length of the data in the container. If the container holds character data that has been converted from one CCSID to another, this is the length of the data *after conversion*.

SET or NODATA option specified

FLENGTH is an output-only field. It must be specified and must be specified as a data-area.

On completion of the retrieval operation, the data area is set to the actual length of the data in the container. If the container holds character data that has been converted from one CCSID to another, this is the length of the data *after conversion*.

INTO(data-area)

specifies the data area into which the retrieved data is to be placed.

INTOCCSID(data-value)

specifies, as a fullword binary number, the Coded Character Set Identifier (CCSID) into which the character data in the container is to be converted. For CICS Transaction Server for z/OS applications, this is typically an EBCDIC CCSID. (However, it is possible to specify an ASCII CCSID if, for example, you want to retrieve ASCII data without it being automatically converted to EBCDIC.)

If INTOCCSID is not specified, its value defaults to the CCSID of the region. The default CCSID of the region is specified on the **LOCALCCSID** system initialization parameter.

Only character data can be converted, and only then if a DATATYPE of CHAR was specified on the **PUT CONTAINER** command used to place the data in the container.

For more information about data conversion with channels, see the *CICS Application Programming Guide*.

For an explanation of CCSIDs, and a list of the CCSIDs supported by CICS, see the *CICS Family: Communicating from CICS on System/390* manual.

NODATA

specifies that no data is to be retrieved. Use this option to discover the length of the data in the container (returned in FLENGTH).

The length of character data may change if data conversion takes place. Therefore, if character data is to be converted into any CCSID *other than that of this region*, when you specify NODATA you should also specify INTOCCSID. This ensures that the correct length of the converted data is returned in FLENGTH.

SET(ptr-ref)

specifies a data area in which the address of the retrieved data is returned.

The data area is maintained by CICS until any of the following occurs:

- A subsequent GET CONTAINER command with the SET option, for the same container in the same channel, is issued by any program that can access this storage.
- The container is deleted by a DELETE CONTAINER command.
- The container is moved by a MOVE CONTAINER command.
- The channel goes out of program scope.

Beware of linking to other programs that might issue one of the above commands.

Do *not* issue a FREEMAIN command to release this storage.

If your application needs to keep the data it should move it into its own storage.

MOVE CONTAINER (CHANNEL):

Move a container (and its contents) from one channel to another.

```
MOVE CONTAINER (CHANNEL)

►►──MOVE──CONTAINER(data-value)──AS(data-value)─────────────────────────►
                                              └─CHANNEL(data-value)─┘

►──────────────────────────────────────────────────────────────────►◄
   └─TOCHANNEL(data-value)─┘
```

Conditions: CHANNELERR, CONTAINERERR, INVREQ

This command is threadsafe.
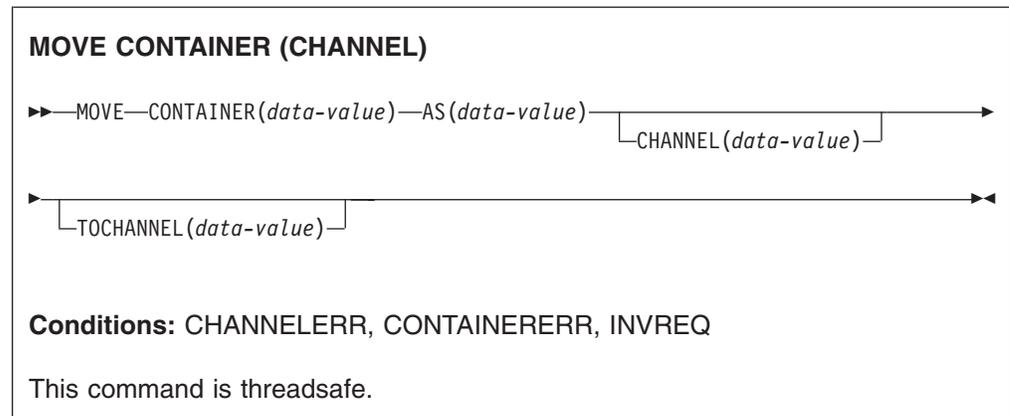
Description

MOVE CONTAINER (CHANNEL) moves a container from one channel to another. After the move, the source container no longer exists.

The source and target containers are identified by name and by the channels that own them. The channel that owns the source container can be identified:
- Explicitly, by specifying the CHANNEL option.
- Implicitly, by omitting the CHANNEL option. If this is omitted, the current channel is implied.

Similarly, the channel that owns the target container can be identified:
- Explicitly, by specifying the TOCHANNEL option.
- Implicitly, by omitting the TOCHANNEL option. If this is omitted, the current channel is implied.

You can move a container:
- From one channel to another.
- Within the same channel—for example, from the current channel to the current channel. This has the effect of renaming the container.

You can use MOVE CONTAINER, instead of GET CONTAINER and PUT CONTAINER, as a more efficient way of transferring data between channels.

Note:
1. The source channel must be within the scope of the program that issues the MOVE CONTAINER command.
2. If the target channel does not exist, within the scope of the program that issues the MOVE CONTAINER command, it is created.

3. If the source container does not exist, an error occurs.
4. If the target container does not already exist, it is created. If the target container already exists, its previous contents are overwritten.
5. If you try to overwrite a container with itself, nothing happens. That is, if you specify the same value for the CONTAINER and AS options, and either omit both the CHANNEL and TOCHANNEL options or give them the same value, so that the same channel is specified, the source container is not changed and not deleted. No error condition is raised.

Options

AS(data-value)
specifies the name (1–16 characters) of the target container. If the target container already exists, its contents are overwritten.

The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Container names are always in EBCDIC. The allowable set of characters for container names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if containers are to be shipped between regions, the characters used in naming
\# them should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

CHANNEL(data-value)
specifies the name (1–16 characters) of the channel that owns the source container. If this option is not specified, the current channel is implied.

CONTAINER(data-value)
specifies the name (1–16 characters) of the source container that is to be moved.

TOCHANNEL(data-value)
specifies the name (1–16 characters) of the channel that owns the target container. If you are specifying a new channel, remember that the acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if channels are to be shipped between regions, the characters used in naming
\# them should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

If this option is not specified, the current channel is implied.

PUT CONTAINER (CHANNEL):

Place data in a named channel container.
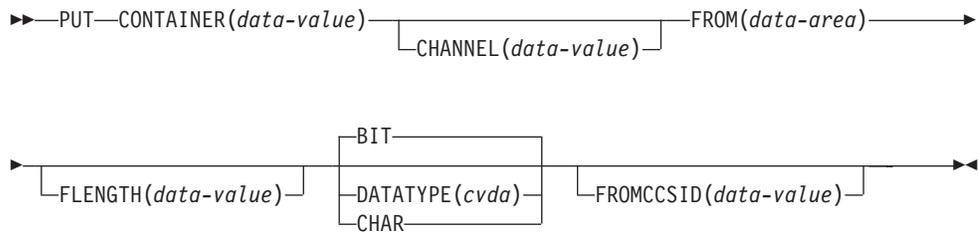
```
┌─────────────────────────────────────────────────────────────────────────┐
│  PUT CONTAINER (CHANNEL)                                                  │
│                                                                           │
│  ►►─PUT─CONTAINER(data-value)──────────────────────FROM(data-area)─────►  │
│                            └─CHANNEL(data-value)─┘                        │
│                                                                           │
│                                 ┌─BIT─┐                                   │
│  ►──────────────────────────────┼─────┼──────────────────────────────►◄  │
│      └─FLENGTH(data-value)─┘     ├─DATATYPE(cvda)─┤  └─FROMCCSID(data-value)─┘ │
│                                 └─CHAR─┘                                  │
│                                                                           │
│  Conditions: CCSIDERR, CHANNELERR, CONTAINERERR, INVREQ,                  │
│  LENGERR                                                                  │
│                                                                           │
│  This command is threadsafe.                                             │
└─────────────────────────────────────────────────────────────────────────┘
```

Description

PUT CONTAINER (CHANNEL) places data in a container associated with a specified channel.

The container is identified by name. The channel that owns the container can be identified:

- Explicitly, by specifying the CHANNEL option.
- Implicitly, by omitting the CHANNEL option. If this is omitted, the current channel is implied.

Note:

1. There is no limit to the number of containers that can be associated with a channel.
2. The size of individual containers is limited only by the amount of storage available.

   CAUTION:
   Take care not to create so many large containers that you limit the amount of storage available to other applications.
3. If the named container does not already exist, it is created. If the named container already exists, its previous contents are overwritten.
4. If the named channel does not already exist, it is created.

Options

CHANNEL(data-value)
specifies the name (1–16 characters) of the channel that owns the container. The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that,

if channels are to be shipped between regions, the characters used in naming them should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

CONTAINER(`data-value`)
specifies the name (1–16 characters) of the container into which data is to be placed.

The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Do not use container names beginning with "DFH", unless requested to do so by CICS.

Container names are always in EBCDIC. The allowable set of characters for container names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if containers are to be shipped between regions, the characters used in naming them should be restricted to A-Z 0-9 & : = , ; < > . - and _.

DATATYPE(`cvda`)
specifies the type of data to be put into the container. This option applies only to *new* containers. If the container already exists its data-type was established when it was created and cannot be changed. CVDA values are:

BIT Bit data. The data in the container cannot be converted. This is the default value, unless FROMCCSID is specified.

CHAR Character data. The data in the container is converted (if necessary) to the code page of the application that *created* the channel. If the channel was created by a client application on an ASCII-based system, this will be an ASCII code page. If it was created by a CICS Transaction Server for z/OS application, it will be an EBCDIC code page. Conversion is only necessary if the client and server programs run on different platforms.

All the data in a container is converted as if it were a single character string. For SBCS code pages, a structure consisting of several character fields is equivalent to a single-byte character string. However, for DBCS code pages this is not the case. If you use DBCS code pages, to ensure that data conversion works correctly you must put each character string into a separate container.

For CHAR containers, the data is stored in the Coded Character Set Identifier (CCSID) specified on the original PUT CONTAINER command that created the container. If the FROMCCSID option was not specified on the original PUT CONTAINER command, the data is stored in the region's default CCSID (or, for CICS-created channels, in the CCSID of the channel). The data on all future PUT CONTAINER CHANNEL commands for this container is converted into this same CCSID. If you want to avoid this, the application program should delete the existing container before issuing the new PUT CONTAINER command, thus recreating the container.

A DATATYPE of CHAR must be specified if the container contains character data *and* the channel will be passed from CICS Transaction Server for z/OS to an ASCII system. If the container contains binary data, or the channel will not be passed to an ASCII system, DATATYPE is an optional parameter.

It is not possible to change the data-type of an existing container by means of a PUT CONTAINER command. For example, if a container is created with a data-type of BIT and a subsequent PUT CONTAINER command specifies a data-type of CHAR, for the same container, an INVREQ condition is raised. If you do need to replace an existing container by one of a different data-type, you must first explicitly delete the existing container.

For more information about data conversion with channels, see the *CICS Application Programming Guide*.

FLENGTH(data-value)
> specifies, as a fullword binary value, the length of the data area from which data is to be read.

FROM(data-area)
> specifies the data area from which the data is written to the container.

FROMCCSID(data-value)
> specifies, as a fullword decimal number, the current Coded Character Set Identifier (CCSID) of the character data to be put into the container. For CICS Transaction Server for z/OS applications, this is typically an EBCDIC CCSID. (However, it is possible to specify an ASCII CCSID, if you want to pass ASCII data.)
>
> FROMCCSID is effective only on the PUT CONTAINER command that creates the container. This is because, for CHAR containers, the data is stored in the CCSID specified on the original PUT CONTAINER command that created the container. If you want to use a different CCSID, the application program should delete the existing container before issuing the new PUT CONTAINER command, thus recreating the container.
>
> If FROMCCSID is specified, DATATYPE(DFHVALUE(CHAR)) is implied.
>
> If FROMCCSID is not specified, its value defaults to the CCSID of the region (or, for CICS-created channels, the CCSID of the channel). The default CCSID of the region is specified on the LOCALCCSID system initialization parameter.
>
> For an explanation of CCSIDs, and a list of the CCSIDs supported by CICS, see the *CICS Family: Communicating from CICS on System/390* manual.

START CHANNEL:

Start a task, passing it a channel.

START CHANNEL

```
►►──START──TRANSID(name)──CHANNEL(name)─────────────────────────────►
                                         ├─TERMID(name)──────┤
                                         └─USERID(data-value)─┘

►─────────────────────────────────────────────────────────────────►◄
  └─SYSID(systemname)─┘
```

Conditions: CHANNELERR, INVREQ, ISCINVREQ, NOTAUTH, RESUNAVAIL, SYSIDERR, TERMIDERR, TRANSIDERR, USERIDERR
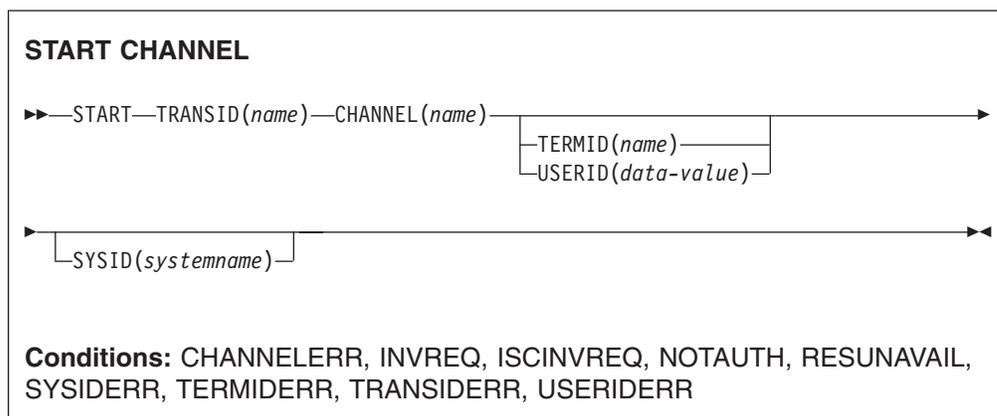
Description

START CHANNEL starts a task, on a local or remote system, passing it a channel.

Typically, the starting task uses the channel to pass data to the started task (although in some circumstances the channel may be empty—see the description of the CHANNEL option). The starting task may also specify a terminal to be used by the started task as its principal facility.

The started task can, for example:
1. Use an ASSIGN CHANNEL command to discover the name of the channel it's been passed
2. Use STARTBROWSE CONTAINER CHANNEL and GETNEXT CONTAINER commands to browse the containers in the channel
3. Use GET CONTAINER CHANNEL commands to access the data in the containers

Some constraints have to be satisfied before the transaction to be executed can be started, as follows:
- If the TERMID option is specified, the named terminal must exist and be available. If the named terminal does not exist, the START is discarded.
- START CHANNEL does not support IMS—that is, you cannot use START CHANNEL to start a transaction on a remote IMS system.

Each START CHANNEL command results in a separate task being started.

Dynamically routed transactions started by START commands

Some transactions started by a subset of START commands can be dynamically routed to a remote region.

START failures without exception conditions

There are some circumstances in which a START command is executed without error, but the started task never takes place:
- When the transaction or its initial program is disabled at the time CICS attempts to create the task.
- When the START specifies a terminal that is not defined (and cannot be located by the XICTENF or XALTENF exits) at the time CICS attempts to create the task.
- You get a TERMIDERR condition if the requested terminal does not exist at the time of the START. However, if the terminal becomes unavailable subsequently, as occurs if the user logs off, your START request is discarded and no TERMIDERR occurs.

These exposures result from the delay between the execution of the START and the time of task creation. Even on a START CHANNEL request, when the START is always immediate, CICS may delay creating the task, either because the required terminal is not free or because of other system constraints.

You can use INQUIRE commands to ensure that the transaction and program are enabled at the time of the START command, but either may become disabled before task creation.

Options

CHANNEL(*name***)**
> specifies the name (1–16 characters) of a channel that is to be made available to the started task. The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

> Channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if channels are to be shipped between regions, the characters used in naming them should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

> The program that issues the START command may:
> * Have created the channel by means of one or more PUT CONTAINER CHANNEL commands
> * Specify its current channel, by name
> * Name a non-existent channel, in which case a new, empty, channel is created

> The started task is given a *copy* of the channel's containers (and the data they contain). The copy is made when the START command is issued.

SYSID(*systemname***)**
> specifies the name of the system to which the request is directed.

TERMID(*name***)**
> specifies the symbolic identifier (1–4 alphanumeric characters) of the principal facility associated with a transaction to be started as a result of a START command. This principal facility can be either a terminal (the usual case) or an APPC session. Where an APPC session is specified, the connection (or modeset) name is used instead of a terminal identifier. This option is required when the transaction to be started must communicate with a terminal; it should be omitted otherwise.

> The terminal identifier must be defined as either a local or a remote terminal on the system in which the START command is executed.

TRANSID(*name***)**
> specifies the symbolic identifier (1–4 characters) of the transaction to be executed by a task started as the result of a START command.

> If SYSID is specified, and names a remote system, the transaction is assumed to be on that system irrespective of whether or not the transaction definition is defined as remote in the PCT. Otherwise the transaction definition is used to find out whether the transaction is on a local or a remote system.

USERID(*data-value***)**
> Specifies the userid under whose authority the started transaction is to run, if the started transaction is not associated with a terminal (that is, when TERMID is not specified). This is referred to as *userid1*.

> If you omit both TERMID and USERID, CICS uses instead the userid under which the transaction that issues the START command is running. This is referred to as *userid2*.

> By using either *userid1* or *userid2* CICS ensures that a started transaction always runs under a valid userid, which must be authorized to all the resources referenced by the started transaction.

CICS performs a surrogate security check against *userid2* to verify that this user is authorized to *userid1*. If *userid2* is not authorized, CICS returns a NOTAUTH condition. The surrogate check is not done here if USERID is omitted.

Modified API commands

The following application programming commands have been modified:

EXEC CICS ASSIGN

The CHANNEL option is added and the STARTCODE option changed:

Options

> **CHANNEL(***data-area***)**
>> Returns the 16-character name of the program's current channel, if one exists; otherwise blanks.
>
> **STARTCODE(***data-area***)**
>> Returns a 2-character value indicating how the transaction that issued the request was started. Changed values are:
>>
>> **Code** **Transaction started by**
>>
>> **S**   START command that did not pass data in the FROM option. It may or may not have passed a channel.
>>
>> **SD**   START command that passed data in the FROM option.

EXEC CICS DELETE CONTAINER (BTS)

There are no syntax changes. The description has been changed to emphasize that the command applies only to BTS, and not channel, containers.

EXEC CICS ENDBROWSE CONTAINER

There are no syntax changes, but the command can now be used with channel, as well as BTS, containers.

EXEC CICS GET CONTAINER (BTS)

There is a new RESP2 value on the INVREQ condition:

Conditions

> **INVREQ**
>> RESP2 values:
>>
>> **2**   The INTOCCSID option was specified without the CHANNEL option, and there is no current channel (because the program that issued the command was not passed one.) INTOCCSID is valid only on GET CONTAINER commands that specify (explicitly or implicitly) a channel. It is not valid on GET CONTAINER (BTS) commands.

EXEC CICS GETNEXT CONTAINER

There are no syntax changes, but the command can now be used with channel, as well as BTS, containers.

EXEC CICS HANDLE ABEND

The program specified to handle the abend is passed the current channel, if one exists; or the communications area (COMMAREA), if one has been established. Previously, it could be passed only the COMMAREA.

EXEC CICS LINK PROGRAM

The CHANNEL option and the CHANNELERR condition are added:

Options

CHANNEL(*name***)**

specifies the name (1–16 characters) of a channel that is to be made available to the invoked program. The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if channels are to be shipped between regions, the characters used in naming them should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

The program that issues the LINK command may:
* Have created the channel by means of one or more PUT CONTAINER CHANNEL commands
* Specify its current channel, by name
* Name a non-existent channel, in which case a new, empty, channel is created

Conditions

CHANNELERR

RESP2 values:

1   The name specified on the CHANNEL option contains an illegal character or combination of characters.

EXEC CICS MOVE CONTAINER (BTS)

There are no syntax changes. The description has been changed to emphasize that the command applies only to BTS, and not CHANNEL, containers.

EXEC CICS PUT CONTAINER (BTS)

There are two new RESP2 values on the INVREQ condition.

Conditions

INVREQ

RESP2 values:

1   The DATATYPE option was specified without the CHANNEL option, and there is no current channel (because the program that issued the command was not passed one.) DATATYPE is valid only on PUT CONTAINER commands that specify (explicitly or implicitly) a channel. It is not valid on PUT CONTAINER (BTS) commands.

2   The FROMCCSID option was specified without the CHANNEL option, and there is no current channel (because the program that issued the command was not passed one.) FROMCCSID is valid only on PUT CONTAINER commands that specify (explicitly or implicitly) a channel. It is not valid on PUT CONTAINER (BTS) commands.

EXEC CICS RETURN

The CHANNEL option and the CHANNELERR condition are added:

Options

CHANNEL(*name***)**

specifies the name (1–16 characters) of a channel that is to be made

available to the next program that receives control. The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if a channel is to be shipped between regions (that is, if the transaction named on the TRANSID option is remote), the characters used in naming it should be restricted to A-Z a-z 0-9 & : = , ; < > . - and _.

The program that issues the RETURN command may:
- Have created the channel by means of one or more PUT CONTAINER CHANNEL commands
- Specify its current channel, by name
- Name a non-existent channel, in which case a new, empty, channel is created

This option is valid only on a RETURN command issued by a program at the highest logical level; that is, a program returning control to CICS.

Conditions

CHANNELERR
RESP2 values:

1   The name specified on the CHANNEL option contains an illegal character or combination of characters.

EXEC CICS STARTBROWSE CONTAINER
The CHANNEL option and the CHANNELERR condition are added:

Options

CHANNEL(*data-value*)
specifies the name (1–16 characters) of the channel whose containers are to be browsed. This must be the name of either the current channel or of a channel created by the program that issues the STARTBROWSE CONTAINER command.

If this option is not specified, and the current context is channel, the current channel's containers are browsed.

Conditions

CHANNELERR
RESP2 values:

2   The channel specified on the CHANNEL option could not be found.

EXEC CICS XCTL
The CHANNEL option and the CHANNELERR condition are added:

Options

CHANNEL(*name*)
specifies the name (1–16 characters) of a channel that is to be made available to the invoked program. The acceptable characters are A-Z a-z 0-9 $ @ # / % & ? ! : | " = ¬ , ; < > . - and _. Leading and

embedded blank characters are not permitted. If the name supplied is less than 16 characters, it is padded with trailing blanks up to 16 characters.

Channel names are always in EBCDIC. The allowable set of characters for channel names, listed above, includes some characters that do not have the same representation in all EBCDIC code pages. We therefore recommend that, if channels are to be shipped between regions, the characters used in naming them should be restricted to A-Z 0-9 & : = , ; < > . - and _.

The program that issues the XCTL command may:
- Have created the channel by means of one or more PUT CONTAINER CHANNEL commands
- Specify its current channel, by name
- Name a non-existent channel, in which case a new, empty, channel is created

Conditions

CHANNELERR
RESP2 values:

1   The name specified on the CHANNEL option contains an illegal character or combination of characters.

Changes to the JCICS API

Note: You can use JCICS commands—including channel- and container-related commands—when writing CICS enterprise beans. However, CICS doesn't support the transmission of channels over IIOP request streams. This means that you cannot, for example, pass a channel to an enterprise bean on a remote region.

New JCICS classes
The following new JCICS classes are introduced:
- `com.ibm.cics.server.CCSIDErrorException`
- `com.ibm.cics.server.Channel`
- `com.ibm.cics.server.ChannelErrorException`
- `com.ibm.cics.server.Container`
- `com.ibm.cics.server.ContainerErrorException`
- `com.ibm.cics.server.ContainerIterator`

Modified JCICS classes
The following JCICS classes are changed:
- Program
- StartRequest
- Task
- TerminalPrincipalFacility

Changes to global user exits

Global user exit programs cannot access containers created by application programs. They can, however, create their own channels and pass them to programs which they call.

Minor changes to the following exits are described in the *CICS Customization Guide*:
- XFCAREQ

- XFCAREQC
- XFCREQ
- XFCREQC
- XICEREQ
- XICEREQC
- XNQEREQ
- XNQEREQC
- XPCREQ
- XPCREQC
- XTDEREQ
- XTDEREQC
- XTSEREQ
- XTSEREQC

Changes to task-related user exits

Task-related user exit programs (TRUEs) cannot access containers created by application programs. They can, however, create their own channels and pass them to programs which they call.

Changes to user-replaceable programs

User-replaceable programs cannot access containers created by application code. They can, however, create their own channels and pass them to programs which they call.

The dynamic and distributed routing programs

There are several changes to the DFHDYPDS communications area passed to the dynamic and distributed routing programs:

DYRACMAA (existing field: dynamic routing program)

This field applies to the routing of:

- Terminal-initiated transactions
- Transactions started by terminal-related START commands
- Program-link (DPL) requests

For the routing of these types of request, DYRACMAA contains one of the following:

- If the user application employs a communications area (COMMAREA), the 31-bit address of the application's COMMAREA
- If the user application employs a channel and has created, within the channel, a container named DFHROUTE, the 31-bit address of the DFHROUTE container
- If the user application has no COMMAREA and no DFHROUTE container, null characters

For the routing of all other types of request, DYRACMAA contains null characters.

For the routing of the three types of eligible request listed above, if the user application employs a COMMAREA:

- When your dynamic routing program is invoked for routing (DYRFUNC=0), the address is that of the *input* communications area (if any). Likewise, when your routing program is invoked because of a route-selection error or for notification (DYRFUNC=1 and 3, respectively), the address is that of the input communications area.
- When your routing program is invoked because a previously-routed transaction or link request has terminated normally (DYRFUNC=2), the

address is that of the *output* communications area (if any). Routed applications can use their output communications area to pass information to the dynamic routing program.

When your routing program is invoked because the routed transaction has abended (DYRFUNC=4), the information in the communications area, or in the DFHROUTE container, is not meaningful.

Your routing program can alter the data in any application's communications area, or DFHROUTE container, addressed by DYRACMAA.

DYRACMAA (existing field: distributed routing program; no change)
is not used by the distributed routing program. On invocation, it is set to zeroes.

DYRACMAL (existing field: dynamic routing program)
This field applies to the routing of:
- Terminal-initiated transactions
- Transactions started by terminal-related START commands
- Program-link (DPL) requests

For the routing of these types of request, DYRACMAL contains one of the following numerical values:

- If the user application employs a COMMAREA, the length, in bytes, of the application's COMMAREA

- If the user application employs a channel and has created, within the channel, a container named DFHROUTE, the length, in bytes, of the data in the DFHROUTE container

- If the user application has no COMMAREA and no DFHROUTE container, zero

For the routing of all other types of request, DYRACMAL contains zero.

DYRACMAL (existing field: distributed routing program; no change)
is not used by the distributed routing program. On invocation, it is set to zeroes.

DYRCHANL (new field)
is the name of the channel, if any, associated with the program-link or START command. This field applies only to the routing of DPL requests, non-terminal-related START requests, and transactions started by terminal-related START requests. For other types of request, or if there is no channel associated with the command, this field contains blanks.

Note that the routing program is given the *name* of the channel, not its address, and so is unable to use the contents of this field to inspect or change the contents of the containers. For information about how the routing program can inspect or change the contents of the application's containers, see "Dynamic routing with channels" on page 221 and the description of the DYRACMAA field.

DYRLEVEL (existing field)
is the level of CICS required in the target AOR to successfully process the routed request. A new value is added:

X'03' CICS TS for z/OS Version 3.1. Currently, may be set only for:
- DPL requests that have a channel associated with them
- START requests that have a channel associated with them
- Method requests for enterprise beans and CORBA stateless objects

DYRTYPE (existing field)
contains the type of request for which the routing program is invoked. Three new values are added:

9 A program-link request with a channel.

The *dynamic* routing program is invoked to route the request.

A   A transaction started by a terminal-related EXEC CICS START command, where there is a channel associated with the START.

The *dynamic* routing program is invoked to route the transaction.

B   A non-terminal-related START request with a channel.

The *distributed* routing program is invoked to route the request.

The meaning of the following existing values has changed. (The changes to the existing descriptions are indicated in **bold**):

2   A transaction started by a terminal-related EXEC CICS START command, where there is no data **and no channel** associated with the START.

3   A transaction started by a terminal-related EXEC CICS START command, where there is data associated with the START **but no channel**.

4   A program-link request **without a channel**.

6   A non-terminal-related START request, with or without data **but with no channel**.

`DYRVER (existing field)`
is the version number of the dynamic routing program interface. **For CICS Transaction Server for z/OS, Version 3 Release 1, the number is "10".**

Changes to monitoring

Some new fields are added to performance-class monitoring records. There is one new group, DFHCHNL, plus additions to the DFHPROG and DFHTASK groups.

All the new fields can be excluded from monitoring records by coding DFHMCT TYPE=RECORD entries in the monitoring control table (MCT).

Performance data in group DFHCHNL
Group DFHCHNL contains the following performance data:

`321 (TYPE-A, 'PGTOTCCT', 4 BYTES)`
The number of CICS requests for channel containers issued by the user task.

`322 (TYPE-A, 'PGBRWCCT', 4 BYTES)`
The number of CICS browse requests for channel containers issued by the user task.

`323 (TYPE-A, 'PGGETCCT', 4 BYTES)`
The number of GET CONTAINER requests for channel containers issued by the user task.

`324 (TYPE-A, 'PGPUTCCT', 4 BYTES)`
The number of PUT CONTAINER requests for channel containers issued by the user task.

`325 (TYPE-A, 'PGMOVCCT', 4 BYTES)`
The number of MOVE CONTAINER requests for channel containers issued by the user task.

`326 (TYPE-A, 'PGGETCDL', 4 BYTES)`
The total length, in bytes, of the data in the containers of all the GET CONTAINER CHANNEL commands issued by the user task.

327 (TYPE-A, 'PGPUTCDL', 4 BYTES)
The total length, in bytes, of the data in the containers of all the PUT CONTAINER CHANNEL commands issued by the user task.

Performance data in group DFHPROG

The following new fields are added to group DFHPROG:

286 (TYPE-A, 'PCDLCSDL', 4 BYTES)
The total length, in bytes, of the data in the containers of all the distributed program link (DPL) requests, with the CHANNEL option, issued by the user task. This total includes the length of any headers to the data.

287 (TYPE-A, 'PCDLCRDL', 4 BYTES)
The total length, in bytes, of the data in the containers of all DPL RETURN CHANNEL commands issued by the user task. This total includes the length of any headers to the data.

306 (TYPE-A, 'PCLNKCCT', 4 BYTES)
Number of local program LINK requests, with the CHANNEL option, issued by the user task.

307 (TYPE-A, 'PCXCLCCT', 4 BYTES)
Number of program XCTL requests issued with the CHANNEL option by the user task.

308 (TYPE-A, 'PCDPLCCT', 4 BYTES)
Number of program distributed program link (DPL) requests issued with the CHANNEL option by the user task.

309 (TYPE-A, 'PCRTNCCT', 4 BYTES)
Number of remote pseudoconversational RETURN requests, with the CHANNEL option, issued by the user task.

310 (TYPE-A, 'PCRTNCDL', 4 BYTES)
The total length, in bytes, of the data in the containers of all the remote pseudoconversational RETURN CHANNEL commands issued by the user task. This total includes the length of any headers to the data.

Performance data in group DFHTASK

The following new fields are added to group DFHTASK:

065 (TYPE-A, 'ICSTACCT', 4 BYTES)
Total number of local interval control START requests, with the CHANNEL option, issued by the user task.

345 (TYPE-A, 'ICSTACDL', 4 BYTES)
Total length, in bytes, of the data in the containers of all the locally-executed START CHANNEL requests issued by the user task. This total includes the length of any headers to the data.

346 (TYPE-A, 'ICSTRCCT', 4 BYTES)
Total number of interval control START CHANNEL requests, to be executed on remote systems, issued by the user task.

347 (TYPE-A, 'ICSTRCDL', 4 BYTES)
Total length, in bytes, of the data in the containers of all the remotely-executed START CHANNEL requests issued by the user task. This total includes the length of any headers to the data.

Changes to statistics

There are new statistics for channel data flowing across a connection. These statistics are mapped by the DFHA14DS DSECT. They are shown in the DFHSTUP "ISC/IRC system entry: Resource statistics" report, and in the DFH0STAT "Connections and Modenames Report".

Table 5. New Fields in the Connections and Modenames Report

Field Heading	Description
Terminal-sharing channel requests	The number of terminal-sharing requests, with channels. Source field: A14ESTTC_CHANNEL
Terminal-sharing channel requests: bytes sent	The number of bytes sent on terminal-sharing channel requests. This is the total amount of data sent on the connection, including any control information. Source field: A14ESTTC_CHANNEL_SENT
Terminal-sharing channel requests: bytes received	The number of bytes received on terminal-sharing channel requests. This is the total amount of data received on the connection, including any control information. Source field: A14ESTTC_CHANNEL_RCVD
Program control LINK requests, with channels	The number of program control LINK requests, with channels, function-shipped across the connection. Source field: A14ESTPC_CHANNEL
LINK channel requests: bytes sent	The number of bytes sent on LINK channel requests. This is the total amount of data sent on the connection, including any control information. Source field: A14ESTPC_CHANNEL_SENT
LINK channel requests: bytes received	The number of bytes received on LINK channel requests. This is the total amount of data received on the connection, including any control information. Source field: A14ESTPC_CHANNEL_RCVD
Interval control START requests, with channels	The number of interval control START requests, with channels, function-shipped across the connection. Source field: A14ESTIC_CHANNEL
START channel requests: bytes sent	The number of bytes sent on START channel requests. This is the total amount of data sent on the connection, including any control information. Source field: A14ESTIC_CHANNEL_SENT
START channel requests: bytes received	The number of bytes received on START channel requests. This is the total amount of data received on the connection, including any control information. Source field: A14ESTIC_CHANNEL_RCVD

Changes to sample programs

Two C language programs from the IIOP Bank Account sample application (transaction BNKQ) have been modified to show how a channel can be passed on EXEC CICS LINK and RETURN commands. DFH$IIBQ, the top-level program, and

DFH$IICC, the program it links to, have been modified to use the channel/container model rather than the COMMAREA model. Comments in the code show the changes that have been made.

Changes to problem determination

Changes to problem determination

Messages

New messages are introduced as a result of enhanced inter-program data transfer. All new and changed messages are described in the *CICS Messages and Codes* manual.

Abend codes

Some new abend codes are introduced, and some existing codes are removed.

New abend codes:

AEYF (CICS TS for z/OS Version 3.1 only):
Explanation

Storage violation by CICS.

A transaction has requested that CICS access a storage area that the transaction itself could not access. This occurred when an invalid storage area was passed to CICS on a PUT CONTAINER or a GET CONTAINER command. The error can occur when:

- Either the FROM or INTO address is specified incorrectly.
- The FLENGTH value specifies a value large enough to cause the area to include storage which the transaction cannot access.

A common cause of this error is specifying the address of a halfword area in the FLENGTH parameter, which expects a fullword area. This error can arise when a program which previously used COMMAREAs (which have halfword lengths) has been modified to use containers (which have fullword lengths).

System Action

The transaction is abnormally terminated with a CICS transaction dump.

User Response

Examine the trace to find the trace entry for entry to DFHEISR, and then identify the parameter in error. If the abend is handled, EXEC CICS ASSIGN ASRASTG, ASRAKEY, ASRASPC, and ASRAREGS give additional information about the abend. At the time of the abend, register 2 points to the storage area at fault.

You will most likely need to do the following:

- Correct the program in error that issued the EXEC CICS PUT CONTAINER or EXEC CICS GET CONTAINER command. Ensure that it supplies the address of a valid storage area and that it supplies an FLENGTH such that no part of the storage area is inaccessible to the transaction. Ensure that FLENGTH refers to a fullword length.

You may also need to consider changing one or more of the following:

- If storage protection is active, change the EXECKEY option, on the CEDA definition of the program that issued the EXEC CICS command, from USER to CICS.
- If storage protection is active, change the TASKDATAKEY attributes on the transaction definition from CICS to USER.
- If transaction isolation is active, change the ISOLATE attribute on the transaction definition from YES to NO.

Module

DFHSRP

AITI (CICS TS for z/OS Version 3.1 only):
Explanation

A mirror transaction processing a START CHANNEL or LINK CHANNEL request has failed while trying to receive data from, or send data to, a connected CICS system. Because a channel may include a considerable amount of data, many calls to terminal control may be required to transmit channel data. DFHMIRS calls program DFHAPCR to perform all the inter-system transmission of channel data. Terminal control has detected an error in one of these calls. The error could be a read time out, or a more serious error in the flows that prevented CICS from correctly processing the data.

System Action

The transaction is terminated. The mirror task is abnormally terminated with a CICS transaction dump.

User Response

If the error was a time out, determine why the remote region has not responded. Examine the trace to determine why the GETMAIN failed. If the CICS region was short on storage, take the necessary steps to correct this. If the region was not short on storage, you may need help from IBM to resolve this problem.

Module

DFHADDRM

AXGA (CICS TS for z/OS Version 3.1 only):
Explanation

Program DFHAPCR has returned an unexpected response. DFHAPCR performs the following functions:
- Extracts the contents of all containers making up a channel and transmits them to a remote system
- Recreates the channel and containers from inbound data received from a remote system

DFHAPCR has either detected an error in inbound data or has received an unexpected response while extracting or recreating channel data.

System Action

The transaction is abnormally terminated with a CICS transaction dump.

User Response

Look for any related CICS messages and abends to determine if there has been a prior failure in Program Manager, which manages containers. Look for exception trace entries from Program Manager or DFHAPCR to determine the cause of the error.

Module

DFHXTP

AXTS (CICS TS for z/OS Version 3.1 only):
Explanation

An attempt was made to pass channel and container data between the transactions in a pseudoconversation, but the next transaction in the pseudoconversation resides in a CICS region that does not support channels and containers.

System Action

The transaction is abnormally terminated with a CICS transaction dump.

User Response

If your application uses channels and containers to pass data between transactions in a pseudoconversation, ensure that all the transactions in the pseudoconversation reside in CICS TS for z/OS Version 3.1, or later, regions.

Module

DFHXTP

AXTU (CICS TS for z/OS Version 3.1 only):
Explanation

Program DFHAPCR has returned an unexpected response. DFHAPCR performs the following functions:
- Extracts the contents of all containers making up a channel and transmits them to a remote system
- Recreates the channel and containers from inbound data received from a remote system

DFHAPCR has either detected an error in inbound data or has received an unexpected response while extracting or recreating channel data.

System Action

The transaction is abnormally terminated with a CICS transaction dump.

User Response

Look for any related CICS messages and abends to determine if there has been a prior failure in Program Manager, which manages containers. Look for exception trace entries from Program Manager or DFHAPCR to determine the cause of the error.

Module

DFHXTP

Other abend codes:
For compatibility purposes, the following new abend codes have been added to
CICS TS for z/OS, Version 2.2 and CICS TS for z/OS, Version 2.3. They do not
apply to CICS TS for z/OS, Version 3.1. If, in the future, you migrate a CICS TS for
z/OS, Version 2.2 or CICS TS for z/OS, Version 2.3 region to CICS TS for z/OS,
Version 3.1, be aware that these abend codes do not occur in CICS TS for z/OS,
Version 3.1.
• AXF9
• AXTT

Trace
The CICS trace points related to the new function are AP 0785 and AP 4E20—AP
4E22.

To control the output of CICS trace information, use CICS trace control in the
normal way.

Migrating from COMMAREAs to channels

Migration of existing functions

• CICS application programs that use traditional communications areas
  (COMMAREAS) to exchange data continue to work as before.
• If you employ a user-written dynamic or distributed routing program for workload
  management, rather than CICSPlex SM, you must modify your program to
  handle the new values that it may be passed in the DYRLEVEL, DYRTYPE, and
  DYRVER fields of the DFHDYPDS communications area—see "The dynamic and
  distributed routing programs" on page 240.

Migration to the new function

This section describes how you can migrate several types of existing application to
use channels and containers rather than communication areas (COMMAREAs).

It's possible to replace a COMMAREA by a channel with a single container. While
this may seem the simplest way to move from COMMAREAs to channels and
containers, it's not good practice to do this.

Also, be aware that a channel may use more storage than a COMMAREA designed
to pass the same data. (See "Designing a channel: best practices" on page 214.)

Because you're taking the time to change your application programs to exploit this
new function, you should implement the "best practices" for channels and
containers—see "Designing a channel: best practices" on page 214. Channels have
several advantages over COMMAREAs (see "Benefits of channels" on page 197)
and it pays to design your channels to make the most of these improvements.

Migrating LINK commands that pass COMMAREAs
To migrate two programs which use a COMMAREA on a LINK command to
exchange a structure, change the instructions shown in Table 6 on page 249.

Table 6. Migrating LINK commands that pass COMMAREAs

Program	Before	After

PROG1	`EXEC CICS LINK PROGRAM(PROG2)`
	`            COMMAREA(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`            CHANNEL(channel-name)`
		`            FROM(structure)`
		`EXEC CICS LINK PROGRAM(PROG2)`
		`            CHANNEL(channel-name)`
		
		`...`
		`EXEC CICS GET CONTAINER(structure-name)`
		`            CHANNEL(channel-name)`
		`            INTO(structure)`

PROG2	`EXEC CICS ADDRESS`
	`     COMMAREA(structure-ptr)`
	`...`
	`RETURN`
		`EXEC CICS GET CONTAINER(structure-name)`
		`            INTO(structure)`
		`...`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`            FROM(structure)`
		`RETURN`

Note: In the COMMAREA example, PROG2, having put data in the COMMAREA, has only to issue a RETURN command to return the data to PROG1. In the channel example, to return data *PROG2 must issue a PUT CONTAINER command before the RETURN*.

Migrating XCTL commands that pass COMMAREAs

To migrate two programs which use a COMMAREA on an XCTL command to pass a structure, change the instructions shown in Table 7.

Table 7. Migrating XCTL commands that pass COMMAREAs

Program	Before	After

PROG1	`EXEC CICS XCTL PROGRAM(PROG2)`
	`            COMMAREA(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`            CHANNEL(channel-name)`
		`            FROM(structure)`
		`EXEC CICS XCTL PROGRAM(PROG2)`
		`            CHANNEL(channel-name)`
		`...`

PROG2	`EXEC CICS ADDRESS`
	`     COMMAREA(structure-ptr)`
	`...`
		`EXEC CICS GET CONTAINER(structure-name)`
		`            INTO(structure)`
		`...`

Migrating pseudoconversational COMMAREAs on RETURN commands

To migrate two programs which use COMMAREAs to exchange a structure as part of a pseudoconversation, change the instructions shown in Table 8.

Table 8. Migrating pseudoconversational COMMAREAs on RETURN commands

Program	Before	After

PROG1	`EXEC CICS RETURN TRANSID(PROG2)`
	`            COMMAREA(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`            CHANNEL(channel-name)`
		`            FROM(structure)`
		`EXEC CICS RETURN TRANSID(TRAN2)`
		`            CHANNEL(channel-name)`

PROG2	`EXEC CICS ADDRESS` `            COMMAREA(structure-ptr)`	`EXEC CICS GET CONTAINER(structure-name)` `            INTO(structure)`

Migrating START data

To migrate two programs which use START data to exchange a structure, change the instructions shown in Table 9 on page 250.

Table 9. Migrating START data

Program	Before	After

PROG1	`EXEC CICS START TRANSID(TRAN2)`
	`            FROM(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`           CHANNEL(channel-name)`
		`           FROM(structure)`
		`EXEC CICS START TRANSID(TRAN2)`
		`           CHANNEL(channel-name)`

PROG2	`EXEC CICS RETRIEVE INTO(structure)`	`EXEC CICS GET CONTAINER(structure-name)` `           INTO(structure)`

Note that the new version of PROG2 is the same as that in the pseudoconversational example.

Migrating programs that use temporary storage to pass data

In previous releases, because the size of COMMAREAs is limited to 32K and channels were not available, some applications used temporary storage queues (TSQs) to pass more than 32K of data from one program to another. Typically, this involved multiple writes to and reads from a TSQ.

If you migrate one of these applications to use channels, be aware that:

- If the TS queue used by your existing application is in main storage, the storage requirements of the new, migrated, application are likely to be similar to those of the existing application.
- If the TS queue used by your existing application is in auxiliary storage, the storage requirements of the migrated application are likely to be greater than those of the existing application. This is because container data is held in storage rather than being written to disk.

Migrating dynamically-routed applications

EXEC CICS LINK and **EXEC CICS START** commands, which can pass either COMMAREAs or channels, can be dynamically routed.

When a LINK or START command passes a COMMAREA rather than a channel, the routing program can, depending on the type of request, inspect or change the COMMAREA's contents. For LINK requests and transactions started by terminal-related START requests (which are handled by the *dynamic* routing program) but not for non-terminal-related START requests (which are handled by the *distributed* routing program) the routing program is given, in the DYRACMAA field of its communication area, the *address* of the application's COMMAREA, and can inspect and change its contents.

Note: The routing program's communication area is mapped by the DFHDYPDS DSECT.

If you migrate a dynamically-routed **EXEC CICS LINK** or START command to use a channel rather than a COMMAREA, the routing program is passed, in the DYRCHANL field of DFHDYPDS, the name of the channel. Note that the routing program is given the *name* of the channel, not its address, and so is unable to use the DYRCHANL field to inspect or change the contents of the channel's containers.

To give the routing program the same kind of functionality with channels, an application that uses a channel can create, within the channel, a special container named DFHROUTE. If the application issues a LINK or terminal-related START request (but not a non-terminal-related START request) that is to be dynamically

routed, the dynamic routing program is given, in the DYRACMAA field of DFHDYPDS, the address of the DFHROUTE container, and can inspect and change its contents.

If you are migrating a program to pass a channel rather than a COMMAREA, you could use its existing COMMAREA structure to map DFHROUTE.

For introductory information about dynamic and distributed routing, see the *CICS Intercommunication Guide*. For information about writing a dynamic or distributed routing program, see the *CICS Customization Guide*.

Coexistence

Coexistence with other CICS products

A CICS TS 3.1 program can invoke a program on a remote CICS region and pass it a channel. For this to work successfully, the remote region must also be at the CICS TS 3.1 level.

Although pre-CICS TS 3.1 regions do not support channels, you can get them to tolerate channels by applying an APAR. By "tolerate" we mean that, if the back-level CICS region is passed a channel, it will return a meaningful abend code.

If a CICS TS 3.1 application tries to send a channel to a back-level region to which the appropriate APAR has been applied, the 3.1 transaction abends with a meaningful abend code. If a CICS TS 3.1 application tries to send a channel to a back-level region to which the appropriate APAR has *not* been applied, the results are unpredictable.

The following list shows the back-level CICS products that tolerate channels, with the APAR that must be applied in each case:

CICS Transaction Server for z/OS, Version 2 Release 3
> APAR PQ92437

CICS Transaction Server for z/OS, Version 2 Release 2
> APAR PQ92437

CICS Transaction Server for OS/390®, Version 1 Release 3
> APAR PQ93048

CICS Transaction Sever for VSE/ESA Release 1.1
> APAR PQ83049

CICSPlex SM support

The CICSPlex SM routing program, EYU9XLOP, has been modified to handle the new values that may be passed in the DYRLEVEL, DYRTYPE, and DYRVER fields of its communications area—see "The dynamic and distributed routing programs" on page 240. When invoked to route any of the following types of request, if there is a channel associated with the request EYU9XLOP routes the request to a CICS TS 3.1 region, if one is available:
* A transaction started by a terminal-related START command
* A non-terminal-related START request
* A program-link request

Changes to CICSPlex SM application programming interface

Changes have been made to the following :

- "TASK resource table"
- "CONNECT resource table" on page 253

TASK resource table

The TASK resource table has been extended to include the following new monitor and statistics attributes:

TMRPGCTC
    Total number of channel container commands

TMRPGBCC
    Number of Browse channel container commands

TMRPGGCC
    Number of GET channel container commands

TMRPGPCC
    Number of PUT channel container commands

TMRPGMCC
    Number of MOVE channel container commands

TMRPGGCL
    Total length in bytes of the data in the containers of all the GET CONTAINER CHANNEL commands

TMRPGPCL
    Total length in bytes of the data in the containers of all the PUT CONTAINER CHANNEL commands

TMRPGCCC
    number of containers created for channel containers

TMRPCDLL
    Total length in bytes of the data in the containers of all the DPL requests with the CHANNEL option

TMRPCDRL
    Total length in bytes of the data in the containers of all DPL RETURN CHANNEL commands

TMRPCLCC
    Number of program LINK requests with CHANNEL option

TMRPCXCC
    Number of program XCTL requests with CHANNEL option

TMRPCDCC
    Number of program distributed program link (DPL) requests with the CHANNEL option

TMRPCRCC
    Number of pseudoconversational RETURN requests with the CHANNEL option

TMRPCRCL
    Total length in bytes of the data in the containers of all the pseudoconversational RETURN CHANNEL commands

TMRICSCC

Total number of local interval control START requests issued with the CHANNEL option

TMRICSCD

Total length, in bytes, of the data in the containers of all local interval control START requests with the CHANNEL option

TMRICSRC

Total number of remote interval control START requests issued with the CHANNEL option

TMRICSRD

Total length, in bytes, of the data in the containers of all remote interval control START requests with the CHANNEL option

CONNECT resource table

The CONNECT resource table has been extended to include the following new monitor and statistics attributes:

ESTPCCHNL

Number of program control LINK requests, with channels, for function shipping.

ESTPCCHNSENT

Number of bytes sent on LINK channel requests. This is the total amount of data sent on the connection, including any control information.

ESTPCCHNRCVD

Number of bytes received on LINK channel requests. This is the total amount of data received on the connection, including any control information.

ESTICCHNL

Number of interval control START requests, with channels, for function shipping.

ESTICCHNSENT

Number of bytes sent on START channel requests. This is the total amount of data sent on the connection, including any control information.

ESTICCHNRCVD

Number of bytes received on START channel requests. This is the total amount of data received on the connection including any control information.

ESTTCCHNL

Number of terminal-sharing channel requests.

ESTTCCHNSENT

Number of bytes sent on terminal-sharing channel requests. This is the total amount of data sent on the connection, including any control information.

ESTTCCHNRCVD

Number of bytes received on terminal-sharing channel requests. This is the total amount of data received on the connection, including any control information.

Changes to CICSPlex SM Web User Interface

New WUI views

The following WUI views have been introduced:
* "Channel usage view"
* "Function ships view"

Channel usage view

There is a new detailed view in the **Active task** view set called **Channel usage**, which is associated with the TASK resource table.

To open the **Channel usage** view, do the following:

1. Click **Active tasks** from the Main menu
2. Select a **Task ID** to open the **Active task** detailed view
3. Scroll down and click **Channel usage**

The new Active task view is displayed. The view name for this Active task is EYUSTARTTASK.DETAIL10. See "TASK resource table" on page 252 for attribute details of the fields displayed in the **Channel usage** view.

Function ships view

There is a new detailed view in the **ISC/MRO connections** view set called **Function ships**, which is associated with the CONNECT resource table.

To open the **Function ships** view, do the following:

1. Click on **CICS operations views** from the main menu
2. Click **Connection operations views**
3. Select **ISC and MRO connections**
4. Click on a Connection ID to open the **ISC/MRO connections** detailed view
5. Scroll down and click **Function ships** to open a new ISC/MRO connections view

The view name for this ISC/MRO connections is EYUSTARTCONNECT.DETAIL3. See "CONNECT resource table" on page 253 for attribute details of the new fields displayed in the **Function ships** detailed view.

Chapter 8. OPENAPI Support

CICS extends the use of Open Transaction Environment (OTE) functionality by providing support for OPENAPI application programs. Prior to this, OPENAPI function was available only to task related user exits (TRUEs).

OPENAPI support allows an application not only to define itself as threadsafe, (meaning it is capable of running on any TCB that CICS deems suitable, either the QR TCB, or an open TCB) but more than that, namely that the application must run on an OPEN TCB rather than on the QR TCB.

The use of OPENAPI programs allows application workloads to be moved off the QR TCB onto multiple open TCBs. If you choose to use OPENAPI programs as a way of running workloads using other (non CICS) APIs remember that the **use of other (non CICS) APIs within CICS is entirely at the discretion and risk of the user. No testing of other (non CICS) APIs within CICS has been undertaken and use of such APIs is not supported by IBM Service.**

In either case you must also be aware that OPENAPI programs still have obligations to the CICS system as a whole. See the *CICS Application Programming Guide*.

A new keyword (API) on the PROGRAM resource definition which takes one of two values CICSAPI or OPENAPI, where CICSAPI is the default. A setting of API(OPENAPI) mandates a setting of CONCURRENCY(THREADSAFE) meaning the application must be coded to threadsafe standards so its application logic is capable of executing with integrity when executed in parallel on multiple TCBs. CICS will handle the threadsafety aspects of any CICS APIs issued from such programs. The new program option applies to user application programs, PLT programs, user replaceable modules and task related user exits. It is ignored for global user exits.

The difference between a CICSAPI QUASIRENT program, a CICSAPI THREADSAFE program and an OPENAPI THREADSAFE program is explained in terms of where it runs:

- A CICSAPI QUASIRENT program only issues CICS APIs and its application logic is not threadsafe. It always runs on the QR TCB.
- A CICSAPI THREADSAFE program is capable of running on either the QR TCB or an open TCB because its application logic is threadsafe. Such a program runs on the QR TCB until some event moves it to an open TCB. A call to an OPENAPI TRUE, such as a DB2 call, is an example of an event that would move a CICSAPI THREADSAFE program to an open TCB. After transferring to an open TCB, the program remains there until something forces it back to the QR TCB, for example a non threadsafe CICS API call. If this happens the program remains on the QR TCB until something (perhaps another DB2 call) forces it back to the open TCB once more.

  A CICSAPI program only uses CICS APIs which are implemented in a way that is independent of the key of the TCB in use. Applications can run successfully in user key or CICS key irrespective of the key of the TCB. So they can run on the QR TCB, an L8 or an L9 TCB.
- An OPENAPI THREADSAFE program always runs on an open TCB, and does so from the start of the program. It is capable of running on an open TCB because its application logic is threadsafe. If use of a non threadsafe CICS

command forces a switch to QR TCB, then CICS switches back to the open TCB again before returning control to the application.

An OPENAPI program can potentially use other (non CICS) APIs, and such APIs generally require the key of the TCB to match the execution key. Therefore user key programs run on L9 TCBs and CICS key programs run on L8 TCBs.

Use of OPENAPI programs can cause more TCB switching than threadsafe CICSAPI programs because of the requirement for the key of the TCB to be correct for OPENAPI programs, because non-threadsafe CICS calls cause a double TCB switch, and because OPENAPI TRUEs always run in CICS key on an L8 TCB. Therefore, for example, a user key OPENAPI program runs on an L9 TCB but if it makes a DB2 call, CICS switches to an L8 TCB to call DB2, then returns to the L9 for the application.

It is highly recommended that existing user key threadsafe CICS-DB2 applications, which have taken advantage of the performance gains of being able to run on the same TCB as the DB2 call, remain defined as CICSAPI THREADSAFE applications. If other functionality is wanted which requires OPENAPI, a separate program should be used.

Candidate programs for defining as OPENAPI THREADSAFE (assuming their application logic is threadsafe) include:

- Programs which use CICS threadsafe APIs only (to avoid the double TCB switch) or only limited non threadsafe CICS commands
- CICS key CICS-DB2 applications
- CPU intensive programs
- Programs wishing to use other (non CICS) APIs at their own risk

Benefits of OPENAPI Support

The main reason for providing support for OPENAPI programs is to allow you to move application workloads off the QR TCB onto multiple open TCBs. This allows the possibility of better utilization of machine resources to achieve better throughput.

Another reason that you might want to use OPENAPI programs could be to allow the use of other (non CICS) APIs.

Use of other APIs is possible because, if an open TCB is blocked by an operating system wait, then only the single application is affected not the whole of CICS, which would be the case if they executed under the QR TCB. Such OPENAPI programs are not permitted to execute on the QR TCB precisely because of this risk of blocking the TCB by an operating system wait and thus affecting the whole of CICS.

- **Use of other (non CICS) APIs within CICS is entirely at the discretion and risk of the user. No testing of other (non CICS) APIs within CICS has been undertaken and use of such APIs is not supported by IBM Service.**

Requirements

OPENAPI Support has no particular requirements for hardware, software or resource usage beyond the general requirements of this release of CICS.

Changes to CICS externals

Changes to system initialization parameters

OPENAPI Support changes the description of the FORCEQR system initialization parameter to limit its relevance to CICSAPI programs, because it does not apply to OPENAPI programs. For the full text of the revised description of FORCEQR, see the *CICS System Definition Guide*.

OPENAPI Support changes the description of the MAXOPENTCBS system initialization parameter to embrace the requirements of OPENAPI and L9 TCBs. For the full text of the revised description of MAXOPENTCBS, see the *CICS System Definition Guide*.

Changes to resource definition

OPENAPI Support introduces a new attribute API to the PROGRAM resource definition.

API has two possible values CICSAPI and OPENAPI. For the full description of the API attribute, and the revised syntax of the PROGRAM definition, see the *CICS Resource Definition Guide*.

Changes to the application programming interface

This topic deals with:
- Obligations of OPENAPI programs, and
- Restrictions when using EDF

Obligations of OPENAPI programs

An OPENAPI program, although freed from the constraints imposed by the QR TCB, nevertheless does have obligations both to the CICS system as a whole and to future users of the L8 or L9 TCB it is using. An L8 or L9 TCB is dedicated for use by the CICS task to which it is allocated, but once the CICS task has completed, the TCB is returned to the dispatcher-managed pool of such TCBs, provided it is still in a clean state (An unclean TCB in this context means that the task using the L8 or L9 mode TCB suffered an unhandled abend in an OPENAPI program. It does not mean that the program has broken the threadsafe restrictions, which CICS would not detect). Note that the TCB is not dedicated for use by a particular OPENAPI program, but is used by all OPENAPI programs and OPENAPI TRUEs invoked by the CICS task to which the L8 mode TCB is allocated. Also, if an application program invoking an OPENAPI program is coded to threadsafe standards, and defined to CICS as threadsafe, it continues to execute on the L8 mode TCB on return from the program.

Threadsafe restrictions:

An OPENAPI program must not treat the executing open TCB environment in such a way that it causes problems for:
- Application program logic that could run on the open TCB
- OPENAPI TRUEs called by the same task
- Future tasks that might use the open TCB
- CICS management code.

At your own risk, if your OPENAPI program decides to use other (non CICS) APIs, you must be aware of the following:

- When invoking CICS services, or when returning to CICS, an OPENAPI program must ensure it restores the MVS™ programming environment as it was on entry to the program. This includes cross-memory mode, ASC mode, request block (RB) level, linkage stack level, TCB dispatching priority, in addition to cancelling any ESTAEs added.

- At CICS task termination, an OPENAPI program must ensure it leaves the open TCB in a state suitable to be reused by another CICS transaction. In particular, it must ensure that all non-CICS resources acquired specifically on behalf of the terminating task are freed. Such resources might include:
  - Dynamically allocated data sets
  - Open ACBs or DCBs
  - STIMERM requests
  - MVS managed storage
  - ENQ requests
  - Attached subtasks
  - Loaded modules
  - Owned data spaces
  - Added access list entries
  - Name/token pairs
  - Fixed pages
  - Security settings (TCBSENV must be set to zero)

- An OPENAPI program must not use the following MVS system services that will affect overall CICS operation:
  - CHKPT
  - ESPIE
  - QEDIT
  - SPIE
  - STIMER
  - TTIMER
  - XCTL / XCTLX
  - Any TSO/E services.

- An OPENAPI program must not invoke under the L8 or L9 mode TCB a Language Environment program that is using MVS Language Environment services, because L8 and L9 mode TCBs are initialized for Language Environment using CICS services.

Restrictions when using EDF

OPEN TCBs and EDF

Even if your program would normally run using an OPEN TCB (L8, L9, X8, or X9) CEDF forces the program to run on the QR TCB, because CEDF itself is not threadsafe.

Changes to the system programming interface

OPENAPI Support has led to changes to the SPI as follows:

INQUIRE DISPATCHER

The descriptions of ACTOPENTCBS and MAXOPENTCBS in the INQUIRE DISPATCHER command are changed to embrace L9 mode TCBs and to refer to OPENAPI programs. For the full text of the revised descriptions, see the *CICS System Programming Reference*.

INQUIRE EXITPROGRAM

A new value CICSAPI for the APIST option in the INQUIRE EXITPROGRAM command is introduced. CICSAPI is synonymous with the previous value BASEAPI. For the full text of the revised descriptions, see the *CICS System Programming Reference*.

INQUIRE PROGRAM

A new option APIST is added to the INQUIRE PROGRAM command. APIST enables you to specify CICSAPI or OPENAPI. For the full text of the description of APIST, see the *CICS System Programming Reference*.

INQUIRE SYSTEM

The description of FORCEQR in the INQUIRE SYSTEM command is changed to limit its relevance to CICSAPI programs, because it does not apply to OPENAPI programs. For the full text of the revised descriptions, see the *CICS System Programming Reference*.

SET DISPATCHER

The descriptions of MAXOPENTCBS in the SET DISPATCHER command is changed to embrace L9 mode TCBs . For the full text of the revised descriptions, see the *CICS System Programming Reference*.

SET SYSTEM

The description of the CVDA value FORCE for the option FORCEQR in the SET SYSTEM command is changed to limit its relevance to CICSAPI programs, because it does not apply to OPENAPI programs. For the full text of the revised descriptions, see the *CICS System Programming Reference*.

Changes to CEMT

OPENAPI Support has led to the following changes:

CEMT INQUIRE DISPATCHER

The descriptions of ACTOPENTCBS and MAXOPENTCBS in the CEMT INQUIRE DISPATCHER command are changed to embrace L9 mode TCBs and to refer to OPENAPI programs. For the full text of the revised descriptions, see *CICS Supplied Transactions*.

CEMT INQUIRE PROGRAM

A new option APIST is added to the CEMT INQUIRE PROGRAM command. APIST enables you to specify CICSAPI or OPENAPI. For the full text of the description of APIST, see *CICS Supplied Transactions*.

CEMT INQUIRE SYSTEM

The description of FORCEQR in the CEMT INQUIRE SYSTEM command is changed to limit its relevance to CICSAPI programs, because it does not apply to OPENAPI programs. For the full text of the revised descriptions, see *CICS Supplied Transactions*.

CEMT SET DISPATCHER

The descriptions of MAXOPENTCBS in the CEMT SET DISPATCHER command is changed to embrace L9 mode TCBs . For the full text of the revised descriptions, see *CICS Supplied Transactions*.

CEMT SET SYSTEM

The description of the value FORCE for the option FORCEQR in the CEMT SET SYSTEM command is changed to limit its relevance to CICSAPI programs, because it does not apply to OPENAPI programs. For the full text of the revised descriptions, see *CICS Supplied Transactions*.

Chapter 9. XPLink Support

Extra Performance Linkage, (from here on it is abbreviated to XPLink), is a z/OS feature which provides high performance subroutine call and return mechanisms. This results in short and highly optimized execution path lengths.

Object Oriented programming is built upon the concept of sending 'messages' to objects which result in that object performing some actions. The message sending activity is implemented as a subroutine invocation. Subroutines, known as member functions in C++ terminology, are normally small pieces of code. The characteristic execution flow of a typical C++ program is, of many subroutine invocations to small pieces of code. Programs of this nature benefit from the XPLink optimization technology.

MVS has a standard subroutine calling convention which can be traced back to the early days of System/360. This convention was optimized for an environment in which subroutines were more complex, there were relatively few of them, and they were invoked relatively infrequently. Object oriented programming conventions have changed this. Subroutines have become simpler but they are numerous, and the frequency of subroutine invocations have increased by orders of magnitude. This change in the size, numbers, and usage pattern, of subroutines made it desirable that the system overhead involved be optimized. XPLink is the result of this optimization.

Note:

> For z/OS 1.4 and above, and CICS 3.1 and above, the advice here that you **CAN** use the XPLINK compiler option with CICS application programs, overrides z/OS advice to the contrary.
>
> z/OS manuals for C and C++ advise you that the XPLINK compiler option is not available to CICS application programs, because that used to be the case. Although these manuals are now being changed, you may be working with a copy of one of these manuals produced before this change.

XPLink, and the X8 and X9 TCBs

CICS provides support for C and C++ programs compiled with the XPLINK option by using the multiple TCB feature in the CICS Open Transaction Environment (OTE) technology. X8 and X9 mode TCBs are defined to support XPLink tasks in CICS key and USER key respectively. Each instance of an XPLink program uses one X8 or X9 TCB.

To use XPLink, your C or C++ application code must be reentrant and threadsafe. The same code instance can be executing on more than one MVS TCB and, without threadsafe mechanisms to protect shared resources, the execution behavior of application code is unpredictable. **This cannot be too strongly emphasized.**

Writing C and C++ programs, which are to be compiled with the XPLINK option, for the CICS environment

The application developer is expected to do the following to take advantage of CICS XPLink support;

261

- Develop the code, strictly adhering to threadsafe programming principles and techniques
- Compile the C or C++ program with the XPLINK option set on
- Indicate in the PROGRAM resource definition that the program is threadsafe
- Consider the use of CICSVAR in CEEUOPT or in #pragma (see the *CICS Application Programming Guide* for details).

All programs using CICS XPLink support must be re-entrant and threadsafe. Only the application developer can guarantee that the code for a particular application satisfies these requirements.

Passing control between XPLink and non-XPLink objects

Each transfer of control from XPLink objects to non-XPLink objects, or the reverse, causes a switch between the QR TCB and an open TCB, (either an X8 or an X9 TCB). In performance terms, TCB switching is costly, you must take this performance overhead into account.

An XPLink object can invoke a non-XPLink object using either the EXEC CICS interface or the Language Environment interface.

A non-XPLink object can only invoke an XPLink object using the EXEC CICS interface. Use of the Language Environment interface for such invocations is not supported.

Changing CICS definitions to obtain CICS support for objects compiled with the XPLINK option

CICS support for programs compiled with the XPLINK option requires only that you show in the PROGRAM resource definition that the program is threadsafe. This indication, and the XPLink "signature" in the load module, are the only things required to put the task on an X8 or X9 TCB.

In the selection of a suitable TCB for a particular program, XPLink takes precedence over the existence of the OPENAPI value for the API attribute on the PROGRAM resource definition.

Benefits of XPLink Support

XPLink Support provides both performance and functional benefits:
- The performance benefits arise from the optimized subroutine linkage technology.
- The functional benefits arise because
  - you can start developing common modules or DLLs which can be used or invoked by programs running under CICS, under TSO/Batch, under IMS™ or under Unix System Services.
  - C++ developers are also able to more fully utilize the C++ Standard Template Library.

Together these generate greater potential for C/C++ code reusability.

Requirements

XPLink Support has no particular requirements for hardware, software or resource usage beyond the general requirements of this release of CICS.

Programming style

All programs using CICS XPLink support must be re-entrant and thread safe.

Changes to CICS externals

Changes to installation

With support for the XPLINK compiler option for C and C++ programs, there are changes to the way that the libraries required for Language Environment must be defined to CICS.

- The library SCEERUN2 must be defined in both the STEPLIB and DFHRPL concatenations.

For more information, see "Installing CICS support for Language Environment" in the *Installation Guide*. For further information, see the *CICS System Programming Reference*.

Changes to system initialization parameters

There is a new system initialization parameter for XPLink Support. The new parameter is MAXXPTCBS.

`MAXXPTCBS={5∨number}`
    Specifies the maximum number of open X8 and X9 TCBs that can exist concurrently in the CICS region. X8 and X9 are the TCBs that are used to run C and C++ programs which are compiled with the XPLINK compiler option. X8 TCBs are used for programs in CICS key, and X9 mode TCBs are used for programs in user key.

    A CICS task is allowed as many X8 and X9 TCBs as it requires, and these TCBs are only kept until program termination

Changes to resource definition

There are no changes to resource definition for XPLink Support. However you must ensure that programs compiled with the XPLink flag set, have the CONCURRENCY attribute set to THREADSAFE in the corresponding program definition.

Changes to the application programming interface
EXEC CICS HANDLE

C and C++ code is restricted to only use EXEC CICS HANDLE ABEND PROGRAM from all the APIs in the EXEC CICS HANDLE 'family'. This restriction continues with XPLink support and is policed by the C and C++ translators.

Other EXEC CICS restrictions

EXEC CICS RETURN, EXEC CICS XCTL and EXEC CICS SEND PAGE RELEASE cause a direct transfer of control from user code to CICS. In C++ these APIs should be used with great care so that the destructors of C++ objects will still get driven appropriately. Failure to drive object destructors can cause storage to leak, open files to be left open, locks to stay locked, and other problems. It is standard practice in C++ to obtain resources during object construction and to release the resource during object destruction. The destructors of stack allocated objects are automatically invoked when they go out of scope. Some C++ objects may also be statically constructed that is they are constructed before the 'main' function is entered and destroyed when the 'main' function ends or the 'exit' call is invoked.

Multithreading

Because C and C++ CICS application programs which are compiled with the XPLink flag, and meet the other criteria described here, run on their own TCBs, application developers might assume that the full set of C, C++ and POSIX APIs are also available to them. That is not so.

CICS application programs which are written in C or C++ should not make use of multithreading techniques in their application code. **You are advised not to use these techniques.**

By multithreading techniques, we mean those forms of coding that create multiple execution paths within the application, for example the use of fork() statements or of pthreads. They are untested in the CICS environment and are considered by CICS to be unsupported. IBM will not accept any problem reports that might be associated with the use of such techniques.

OPEN TCBs and EDF

Even if your program would normally run using an OPEN TCB (L8, L9, X8, or X9) CEDF forces use of the QR TCB, because CEDF itself is not threadsafe.

Changes to the system programming interface
INQUIRE DISPATCHER

The `EXEC CICS INQUIRE DISPATCHER` command is changed to include the ACTXPTCBS and MAXXPTCBS options.

ACTXPTCBS(value)
>   displays the number of X8 and X9 mode open TCBs that are currently active (that is, allocated to a user task).

MAXXPTCBS(value)
>   displays the maximum number of X8 and X9 mode open TCBs that can exist concurrently in the CICS region. The value can be in the range 1-999. You can reset this value by overtyping it with a different value.

INQUIRE PROGRAM

The `EXEC CICS INQUIRE PROGRAM` command is changed to include the XPLink value for the Runtime option.

`Runtime`
>   displays information about the runtime environment of the program. The new value in this list is:

>   **XPLink**
>>   The program is a C or C++ program which has been compiled using the XPLINK option.

.

SET DISPATCHER

The `EXEC CICS SET DISPATCHER` command is changed to include the MAXXPTCBS option.

MAXXPTCBS(value)
> specifies the maximum number of X8 and X9 mode open TCBs that can exist concurrently in the CICS region. The value specified can be in the range 1 to 999. If you reduce MAXXPTCBS from its previously defined value, and the new value is less than the number of open TCBs currently allocated, CICS detaches TCBs to achieve the new limit only when they are freed by user tasks. Transactions are not abended to allow TCBs to be detached to achieve the new limit. If there are tasks queued waiting for an X8 or X9 mode TCB and you increase MAXXPTCBS from its previously defined value, CICS attaches a new TCB to resume each queued task, up to the new limit.

Changes to CEMT

CEMT INQUIRE DISPATCHER

The CEMT INQUIRE DISPATCHER command is changed to include the ACTXPTCBS and MAXXPTCBS options.

ACTXPTCBS(value)
> displays the number of X8 and X9 mode open TCBs that are currently active (that is, allocated to a user task).

MAXXPTCBS(value)
> displays the maximum number of X8 and X9 mode open TCBs that can exist concurrently in the CICS region. The value can be in the range 1-999. You can reset this value by overtyping it with a different value.

CEMT INQUIRE PROGRAM

The CEMT INQUIRE PROGRAM command is changed to include the XPLink value for the Runtime option.

`Runtime`
> displays information about the runtime environment of the program. The new value in this list is:

> **XPLink**
> > The program is a C or C++ program which has been compiled using the XPLINK option.

.

CEMT SET DISPATCHER

The CEMT SET DISPATCHER command is changed to include the MAXXPTCBS option.

MAXXPTCBS(value)
> specifies the maximum number of X8 and X9 mode open TCBs that can exist concurrently in the CICS region. The value specified can be in the range 1 to 999. If you reduce MAXXPTCBS from its previously defined value, and the new value is less than the number of open TCBs currently allocated, CICS detaches TCBs to achieve the new limit only when they are freed by user tasks. Transactions are not abended to allow TCBs to be detached to achieve the new limit. If there are tasks queued waiting for an X8 or X9 mode TCB and you increase MAXXPTCBS from its previously defined value, CICS attaches a new TCB to resume each queued task, up to the new limit.

Changes to global user exits

XPCFTCH

When the exit XPCFTCH is invoked from a C or C++ programs that was compiled with the XPLINK option, a flag is set indicating that any modified entry point address, if specified by the exit, will be ignored.

XPCTA

When the exit XPCTA is invoked from a C or C++ programs that was compiled with the XPLINK option, a flag is set indicating that a resume address, if specified by the exit, will be ignored.

DFHUEPAR

Two symbolic values, UEPTX8 and UEPTX9, are added to the table of TCB indicators in DFHUEPAR.

Changes to user-replaceable programs

A New user-replaceable program **DFHAPXPO** is provided.

DFHAPXPO is loaded during the PIPI preinitialization phase of each Language Environment enclave where C or C++ programs compiled with the XPLINK option are to be run. It allows you to alter the default Language Environment run-time options. See the *z/OS Version 1.4 Language Environment Programming Guide*, SC22-7561, for details of the Language Environment options that can be reset. The program must be written in Assembler language.

Changes to monitoring

In the topic "A note about wait (suspend) times" in the "CICS Performance Guide", a new item is added to the Table, as follows:

Table 10. Performance class wait (suspend) fields

Field-Id	Group Name	Description
282	DFHTASK	CICS MAXXPTCBS delay time

New fields are added to performance-class monitoring records. These are additions to group DFHTASK, and there are some changes to the description of existing items in that group:

New items

271 (TYPE-S, "X8CPUT", 8 BYTES)
The processor time during which the user task was dispatched by the CICS dispatcher domain on a CICS X8 mode TCB. When a transaction invokes a C or C++ program that was compiled with the XPLINK option, and that is defined with EXECKEY=CICS, it is allocated and uses a CICS X8 mode TCB. (An X8 mode TCB can also be allocated if the program is defined with EXECKEY=USER, but the storage protection facility is inactive.) Once a task has been allocated a X8 mode TCB, that same TCB remains associated with the task until the program completes.

272 (TYPE-S, "X9CPUT", 8 BYTES)
The processor time during which the user task was dispatched by the

CICS dispatcher domain on a CICS X9 mode TCB. When a transaction invokes a C or C++ program that was compiled with the XPLINK option, and that is defined with EXECKEY=USER, it is allocated and uses a CICS X9 mode TCB. (If the storage protection facility is inactive, an X8 mode TCB is used instead of an X9 mode TCB.) Once a task has been allocated an X9 mode TCB, that same TCB remains associated with the task until the program completes

282 (TYPE-S, "MAXXTDLY", 8 BYTES)
The elapsed time in which the user task waited to obtain a CICS XP TCB (X8 or X9 mode), because the CICS system had reached the limit set by the system parameter, MAXXPTCBS. The X8 and X9 mode open TCBs are used exclusively by C and C++ programs that were compiled with the XPLINK option.

Note: This field is a component of the task suspend time field, **SUSPTIME** (group name: DFHTASK, field id: 014).

Changed items

007 (TYPE-S, "USRDISPT", 8 BYTES)
X8, and X9 are added to the list of TCB modes.

008 (TYPE-S, "USRCPUT", 8 BYTES)
X8, and X9 are added to the list of TCB modes.

262 (TYPE-S,'KY8DISPT',8 BYTES)
To the list of items that make up the total elapsed time,

- When a transaction invokes a C or C++ program that was compiled with the XPLINK option, and that is defined with EXECKEY=CICS, it is allocated a CICS X8 mode TCB, and dispatched on that TCB. The TCB remains associated with the task until the program ends.

is added.

263 (TYPE-S,'KY8CPUT',8 BYTES)
To the list of items that make up the processor time,

- When a transaction invokes a C or C++ program that was compiled with the XPLINK option, and that is defined with EXECKEY=CICS, it is allocated a CICS X8 mode TCB, and dispatched on that TCB. The TCB remains associated with the task until the program ends.

is added.

264 (TYPE-S, "KY9DISPT", 8 BYTES)
To the list of items that make up the total elapsed time,

- When a transaction invokes a C or C++ program that was compiled with the XPLINK option, and that is defined with EXECKEY=USER, it is allocated a CICS X9 mode TCB, and dispatched on that TCB. The TCB remains associated with the task until the program ends.

is added.

265 (TYPE-S, "KY9CPUT", 8 BYTES)
To the list of items that make up the processor time,

- When a transaction invokes a C or C++ program that was compiled with the XPLINK option, and that is defined with EXECKEY=USER, it is allocated a CICS X9 mode TCB, and dispatched on that TCB. The TCB remains associated with the task until the program ends.

is added.

All the new fields, and the changed fields, can be excluded from monitoring records by coding DFHMCT TYPE=RECORD entries in the monitoring control table (MCT).

Changes to statistics

There are changes to "Dispatcher domain: TCB Mode statistics" which are mapped by the DFHDSGDS DSECT.

The changes add the X8 and X9 mode TCBs, and the XP pool of TCBs, to the descriptions of the items in this table.

Table 11. Changed Fields in the Dispatcher domain: TCB Mode statistics

DFHSTUP name	Field name	Description
TCB Mode	DSGTCBNM	is the name of the CICS dispatcher TCB mode, either QR, RO, CO, SZ, RP, FO, SL, SO, S8, D2, JM, L8, L9, J8, J9, X8, or X9.. Reset characteristic: not reset
TCB Pool	DSGTCBMP	is the name of the TCB pool in which this CICS dispatcher TCB mode is defined, either N/A, OPEN, JVM, or XP. Reset characteristic: not reset

Migration

Migration of existing functions

No action is needed to continue using C and C++ without XPLink.

Migration to the new function

To take advantage of the support that is now available for the XPLink compiler option for C and C++ programs, consider the following points:

- Ensure that your C or C++ program is reentrant, and threadsafe, or modify it so that it conforms to these standards, see "Migration planning for threadsafe programming and the open transaction environment (OTE)" in the *Migration Guide*
- If your program uses the XPCFTCH or XPCTA exits, take note of the advice in "Global User exits and XPLink" in the *CICS Application Programming Guide* that:
  - CICS disregards any attempt by XPCFTCH to modify the entry point
  - CICS disregards any attempt by XPCTA to define a resume address

  You must find other ways to manage such requirements, or conclude that this program is not a suitable candidate for XPLINK optimization.
- Recompile the program using the XPLINK compiler option.
- Update the concurrency attribute of the PROGRAM resource definition for this program, setting the value to threadsafe.

CICSPlex SM support

There are a number of changes to the CICSPlex SM Web User Interface and API to match the new support for XPLINK C++.

Changes to the CICSPlex SM application programming interface
Changes to resource tables

Changes have been made to the following resource tables:
- "PROGRAM resource table"
- "PROGDEF resource table"
- "TASK resource table"
- "CICSRGN resource table"

PROGRAM resource table

The PROGRAM resource table includes the following attributes:

RUNTIME
　　Shows the CVDA (XPLINK) value 1068

APIST Shows the CVDA value that indicates the API status of CICSAPI or OPENAPI

PROGDEF resource table

The PROGDEF resource table includes the following attribute:

API Shows the EYUDA values of CICSAPI or OPENAPI

TASK resource table

The TASK resource table includes the following attribute:

\# **TMRL9CPU**
\# 　　Shows the user task L9 mode CPU time

CICSRGN resource table

The CICSRGN resource table includes the following SPI attributes:

MAXXPTCBS
　　Shows the current maximum number of TCBs in the XPLink OTE X8/X9 mode pool.

ACTXPTCBS
　　Shows the actual number of TCBs in the XPLink OTE X8/X9 mode pool.

The existing DISPATCHER SPI attributes MAXHPTCBS and ACTHPTCBS are no longer supported and are flagged NOTVALID.

Changes to CICSPlex SM Web User Interface

Changes have been made to the following WUI views:
- "Program view" on page 270
- "Program Definition view" on page 270
- "CPU and TCB information view" on page 270

- "CICS region view"

Program view

The following attribute has been added to the Program
(EYUSTARTPROG.DETAILED) view:

APIST API status

Program Definition view

The following attribute has been added to the Program Definition
(EYUSTARTPROGDEF.DETAILED) view:

API   Application programming interfaces

CPU and TCB information view

The following attribute has been added to the CPU and TCB information view, one
of the Active tasks view set (EYUSTARTTASK.DETAIL9) within the Task operations
view:

TMRL9CPU
        User task L9 mode CPU time

CICS region view

The following attributes have been added to the CICS region
(EYUSTARTCICSRGN.DETAILED) view:

MAXXPTCBS
        Shows the current maximum number of TCBs in the XPLink OTE X8/X9
        mode pool.

ACTXPTCBS
        Shows the actual number of TCBs in the XPLink OTE X8/X9 mode pool.

The existing DISPATCHER SPI attributes MAXHPTCBS and ACTHPTCBS are no
longer supported and are flagged NOTVALID.

Chapter 10. Support for Language Environment conforming assembler MAIN programs

You can now produce assembler MAIN programs which are Language Environment conforming.

Until now, the only way to use Language Environment conforming assembler programs within CICS was to use a call from a COBOL, PLI, or C Language Environment conforming program and linkedit the assembler program with the high-level language (HLL) program. This made the assembler program a Language Environment subroutine. It had to have MAIN=NO on CEEENTRY. The user had to specify NOPROLOG and NOEPILOG and then code the CEEENTRY and CEETERM calls separately. A CICS PROGRAM resource could not be defined as both ASM and LE370.

CICS now supports the coding of Language Environment conforming assembler MAIN programs. A new translator option LEASM causes Language Environment function to be used to set up the program's environment. Such programs must be linkedited with stub DFHELII rather than DFHEAI.

This support also enables use of the Debugger for Assembler programs.

Benefits of Support for Language Environment conforming assembler MAIN programs

Support for Language Environment conforming assembler MAIN programs extends the availability of Language Environment use, and it makes Debugger support available with such programs.

Requirements

Support for Language Environment conforming assembler MAIN programs has no particular requirements for hardware, software or resource usage beyond the general requirements of this release of CICS.

Changes to CICS externals

Changes to resource definition

The Language attribute of the PROGRAM resource definition has changed descriptions of some of the values that can be specified.

Language({COBOL|ASSEMBLER|LE370|PLI})
>   specifies the program language:

>   **ASSEMBLER**
>>   This is an assembler language program which was not translated using the LEASM translator option. LEASM is used to translate those assembler programs which are to be Language Environment-conforming MAIN programs..

>   **LE370** The program exploits multi-language support, has been compiled by a Language Environment-conforming compiler, or it is an assembler MAIN program which was translated using the LEASM option to produce a Language Environment-conforming program.

Changes to the application programming interface

Language restrictions

When programming in assembler and planning to translate your program using the LEASM option, these restrictions are added to those which apply for all assembler programs.

- Register 2 cannot be used as a code base register.
- Register 12 is reserved by Language Environment to point to the Language Environment common anchor area (CAA) and so cannot be used at all by the program without being saved and restored as appropriate.
- Register 13 must be used as the one and only working storage base register.
- The program cannot be a Global User Exit program (GLUE) or a Task-Related User Exit program (TRUE).
- The program must not use, or depend on, any AMODE(24) code.

Translator options

A new translator option LEASM (valid only for assembler programs) allows you to specify that this program is to be translated using the macros that will make it a Language Environment-conforming program, ready for assembly as a MAIN program.

Specification of LEASM results in the setting of a new assembler global &DFHEILE. A fourth positional parameter LE has been added to the DFHEIGBL macro that the translator inserts at the top of every output file. DFHEIGBL changes to set &DFHEILE on if LEASM is specified.

The translator sets a value (X'12') for the language in ARG0 when LEASM has been specified. The output from the translator will be identical to that produced without specifying LEASM in every other way.

If &DFHEILE is set, the DFHEISTG, DFHEIENT, DFHEIRET and DFHEIEND macros expand differently to create an LE environment rather than a normal CICS environment. This allows your programs that have used NOPROLOG and NOEPILOG and coded their own DFHEIENT and other macros to take advantage of Language Environment support without changing their program source. For example, all programs that require more than one code base register fall into this category because the translator does not support multiple code base registers.

DFHEISTG

If &DFHEILE is set, the statement CEEDSA SECTYPE=OS will be added at the top of DFHEISTG, immediately after the DSECT statement. This causes the standard Language Environment DSA to be included, without a DSECT statement.

DFHEIENT

DFHEIENT generates a Language Environment CEEENTRY macro that sets up code addressability and working storage, rather than using the standard CICS methods.

Code addressability

DFHEIENT has a CODEREG parameter that defaults to 3. When the translator option LEASM is specified, the CEEENTRY is generated by DFHEIENT with the BASE parameter a straight copy of the CODEREG value.

CEEENTRY does not allow the use of registers 2 or 12 for code addressability. Register 12 must always contain the address of the Language Environment CAA during execution of a Language Environment program. Register 2 is not generally used as a code base because it is modified by instructions such as TRT. DFHEIENT will generate an error message if Register 2 or Register 12 is specified on CODEREG in a program translated using the LEASM option.

Working storage addressability

DFHEIENT has a DATAREG parameter that defaults to 13. CEEENTRY doesn't allow anything other than R13 for working storage, and does not support multiple working storage registers. DFHEIENT disallows the use of anything other than 13 for DATAREG in a program translated using the LEASM option.

DFHEIRET

Generates CEETERM RC=0 for a program translated using the LEASM option.

EXAMPLE LEASM PROGRAM

Here is a simple CICS assembler program.

```
*ASM XOPTS(LEASM)
DFHEISTG DSECT
OUTAREA  DS    CL200               DATA OUTPUT AREA
*
EIASM    CSECT ,
         MVC  OUTAREA(40),MSG1
         MVC  OUTAREA(4),EIBTRMID
         EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
         EXEC CICS RECEIVE
         MVC  OUTAREA(13),MSG2
         EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
         EXEC CICS RETURN
*
MSG1     DC   C'xxxx: ASM program invoked. ENTER TO END.'
MSG2     DC   C'PROGRAM ENDED'
         END
```

When translated and assembled, it becomes:

```
*ASM XOPTS(LEASM)
        DFHEIGBL ,,,LE             INSERTED BY TRANSLATOR
*,&DFHEIDL; SETB 0  1 MEANS EXEC DLI IN PROGRAM                  01-DFHEI
*,&DFHEIDB; SETB 0  1 MEANS BATCH PROGRAM                        01-DFHEI
*,&DFHEIRS; SETB 0  1 MEANS RSECT                                01-DFHEI
*,&DFHEILE; SETB 1  1 MEANS LE MAIN                              01-DFHEI
DFHEISTG DSECT
        DFHEISTG                   INSERTED BY TRANSLATOR
*************************************************************************
*            EXEC INTERFACE DYNAMIC STORAGE                     *
*************************************************************************
DFHEISTG DSECT                     EXEC INTERFACE STORAGE     @BBAC81A 01-DFHEI
        USING *,DFHEIPLR           ESTABLISH ADDRESSABILITY   @BBAC81A 01-DFHEI
*
*************************************************************************
*  D Y N A M I C   S T O R A G E   A R E A   ( D S A )         *
*************************************************************************
*
CEEDSA    DS    0D        Just keep the same label for formulae    02-CEEDS
*
CEEDSAFLAGS DS XL2                 DSA flags                        02-CEEDS
CEEDSALNGC EQU X'1000'               C library DSA                  02-CEEDS
CEEDSALNGP EQU X'0800'               PL/I library DSA               02-CEEDS
CEEDSAEXIT EQU X'0008'               An Exit DSA                    02-CEEDS
CEEDSAMEMD DS  XL2                 Member defined                   02-CEEDS
CEEDSABKC DS   A                   Addr of DSA of caller            02-CEEDS
CEEDSAFWC DS   A                   Addr of DSA of last called rtn   02-CEEDS
```

```
**********************************************************************
*                                                                    *
* CONTROL BLOCK NAME = DFHEIBLK                                       *
*                                                                    *
* NAME OF MATCHING PL/AS CONTROL BLOCK = None                         *
*                                                                    *
* DESCRIPTIVE NAME = %PRODUCT EXEC Interface Block.                   *
*                                                                    *
*    @BANNER_START                         02                        *
*    Licensed Materials - Property of IBM                            *
*                                                                    *
*    "Restricted Materials of IBM"                                   *
*                                                                    *
*    5697-E93                                                        *
*                                                                    *
*    (C) Copyright IBM Corp. 1990, 1993                              *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*    @BANNER_END                                                     *
*                                                                    *
* STATUS = %XA20                                                      *
*                                                                    *
* FUNCTION = EXEC Interface Block.                                    *
*                                                                    *
*       The exec interface block contains information on the         *
*       transaction identifier, the time and date, and the cursor    *
*       position on a display device. Some of the other fields are   *
*       set indicating the next action that a program should take    *
*       in certain circumstances.                                    *
*       DFHEIBLK also contains information that will be helpful       *
*       when a dump is being used to debug a program.                *
*       This control block is included automatically by an           *
*       application program using the command-level interface.       *
*       EISEIBA in the EIS addresses the EIB.                        *
*                                                                    *
*                                                                    *
*                                                                    *
* NOTES :                                                            *
*    DEPENDENCIES = S/370                                            *
*    MODULE TYPE = Control block definition                          *
*    PROCESSOR = Assembler                                           *
*                                                                    *
*------------------------------------------------------------------- *
*                                                                    *
* CHANGE ACTIVITY :                                                  *
*   £SEG(DFHEIBLK),COMP(COMMAND),PROD(%PRODUCT) :                    *
*                                                                    *
*    PN= REASON REL YYMMDD HDXXIII : REMARKS                         *
*    £L1= 550    %0G 900515 HDFSPC  : Add an EIB length equate       *
*    £D1= I05119 %B1 930226 HDDHDMA : Correct comments for date field *
*    £P1= M60581 %B0 900116 HDAEGB  : Change for PLXMAP to data areas *
*                                                                    *
**********************************************************************
*             EXEC INTERFACE BLOCK                                   *
**********************************************************************
DFHEIBLK DSECT                    EXEC INTERFACE BLOCK     @BBAC81A 01-DFHEI
         USING *,DFHEIBR                                   @BBAC81A 01-DFHEI
```

```
EIBTIME  DS   PL4                  TIME IN 0HHMMSS FORMAT     @BBAC81A 01-DFHEI
EIBDATE  DS   PL4                  DATE IN 0CYYDDD+ FORMAT,       @D1C 01-DFHEI
*                                  where C is the century         @D1A
*                                  indicator (0=1900, 1=2000),    @D1A
*                                  YY is the year, DDD is the     @D1A
*                                  day number and '+' is the      @D1A
*                                  sign byte (positive)           @D1A
EIBTRNID DS   CL4                  TRANSACTION IDENTIFIER     @BBAC81A 01-DFHEI
EIBTASKN DS   PL4                  TASK NUMBER               @BBAC81A 01-DFHEI
EIBTRMID DS   CL4                  TERMINAL IDENTIFIER       @BBAC81A 01-DFHEI
EIBRSVD1 DS   H                    RESERVED                  @BBAC81A 01-DFHEI
EIBCPOSN DS   H                    CURSOR POSITION           @BBAC81A 01-DFHEI
EIBCALEN DS   H                    COMMAREA LENGTH           @BBAC81A 01-DFHEI
EIBAID   DS   CL1                  ATTENTION IDENTIFIER      @BBAC81A 01-DFHEI
EIBFN    DS   CL2                  FUNCTION CODE             @BBAC81A 01-DFHEI
EIBRCODE DS   CL6                  RESPONSE CODE             @BBAC81A 01-DFHEI
EIBDS    DS   CL8                  DATASET NAME              @BBAC81A 01-DFHEI
EIBREQID DS   CL8                  REQUEST IDENTIFIER        @BBAC81A 01-DFHEI
EIBRSRCE DS   CL8                  RESOURCE NAME             @BBDIA0U 01-DFHEI
EIBSYNC  DS   C                    X'FF' SYNCPOINT REQUESTED @BBDIA0U 01-DFHEI
EIBFREE  DS   C                    X'FF' FREE REQUESTED      @BBDIA0U 01-DFHEI
EIBRECV  DS   C                    X'FF' RECEIVE REQUIRED    @BBDIA0U 01-DFHEI
EIBSEND  DS   C                    RESERVED                  @BM13417 01-DFHEI
EIBATT   DS   C                    X'FF' ATTACH RECEIVED     @BBDIA0U 01-DFHEI
EIBEOC   DS   C                    X'FF' EOC RECEIVED        @BBDIA0U 01-DFHEI
EIBFMH   DS   C                    X'FF' FMHS RECEIVED       @BBDIA0U 01-DFHEI
EIBCOMPL DS   C                    X'FF' DATA COMPLETE                01-DFHEI
EIBSIG   DS   C                    X'FF' SIGNAL RECEIVED              01-DFHEI
EIBCONF  DS   C                    X'FF' CONFIRM REQUESTED            01-DFHEI
EIBERR   DS   C                    X'FF' ERROR RECEIVED               01-DFHEI
EIBERRCD DS   CL4                  ERROR CODE RECEIVED                01-DFHEI
EIBSYNRB DS   C                    X'FF' SYNC ROLLBACK REQ'D          01-DFHEI
EIBNODAT DS   C                    X'FF' NO APPL DATA RECEIVED        01-DFHEI
EIBRESP  DS   F                    INTERNAL CONDITION NUMBER          01-DFHEI
EIBRESP2 DS   F                    MORE DETAILS ON SOME RESPONSES     01-DFHEI
EIBRLDBK DS   CL1                  ROLLED BACK                        01-DFHEI
*
EIBLENG  EQU  *-EIBTIME            Length of EIB                  @L1A 01-DFHEI
***********************************************************************
*            END OF EXEC INTERFACE BLOCK                            *
***********************************************************************
DFHEIBR  EQU  11                   EIB REGISTER              @BA02936 01-DFHEI
                                                                @01A  02-CEEEN
```

```
    ***********************************************************************
    *              PROLOG CODE FOR EXEC INTERFACE                        *
    ***********************************************************************
    *&DFHEICS; CEEENTRY PPA=DFHPPA,MAIN=YES,PLIST=OS,
    *              BASE=&CODEREG;,
    *              AUTO=(DFHEIEND-DFHEISTG)
    TESTLE  CSECT ,                                                     02-CEEEN
    TESTLE  RMODE ANY                                                   02-CEEEN
    TESTLE  AMODE ANY                                                   02-CEEEN
            ENTRY TESTLE                                                02-CEEEN
            PUSH  USING                                                 02-CEEEN
            DROP  ,                                             @02A    02-CEEEN
            USING *,15                                                  02-CEEEN
            B     CEEZ0007                                              02-CEEEN
            DC    X'00C3C5C5'                                           02-CEEEN
    CEEY0007     DC    A((((DFHEIEND-DFHEISTG)+7)/8)*8)               X02-CEEEN
                                        . Size of automatic storage.
            DC    A(DFHPPA-TESTLE)   . Address of PPA for this program   02-CEEEN
            B     1(,15)                                                02-CEEEN
    CEEZ0007    EQU *                                                   02-CEEEN
            STM   14,12,CEEDSAR14-CEEDSA(13)                            02-CEEEN
            L     2,CEEINPL0007                        5@01D @01C       02-CEEEN
            L     15,CEEINT0007                              @01C       02-CEEEN
            DROP  15                                         @01A       02-CEEEN
            BALR  14,15                                                 02-CEEEN
            LR    2,1                                                   02-CEEEN
            L     14,752(,12)                                           02-CEEEN
            OI    8(14),X'80'                                           02-CEEEN
            BALR  3,0                                        @01A       02-CEEEN
            USING *,3                                                   02-CEEEN
            L     3,CEEOEPV0007                              @01A       02-CEEEN
            POP   USING                                      @01A       02-CEEEN
            USING TESTLE,3                                   @01A       02-CEEEN
            L     1,CEEDSANAB-CEEDSA(,13)  Get the current NAB          02-CEEEN
            L     0,CEEY0007                                            02-CEEEN
            ALR   0,1                      Compute new value.           02-CEEEN
            CL    0,CEECAAEOS-CEECAA(,12)  Compare with EOS.            02-CEEEN
            BNH   CEEX0007                                              02-CEEEN
            L     15,CEECAAGETS-CEECAA(,12) Get address overflow routine 02-CEEEN
            BALR  14,15                     Get another stack segment.   02-CEEEN
            LR    1,15                                                  02-CEEEN
            B     CEEX0007                 Branch around statics  @01A  02-CEEEN
    CEEINPL0007    DC    A(CEEINPL)                            @01A     02-CEEEN
    CEEINT0007     DC    V(CEEINT)                             @01A     02-CEEEN
    CEEOEPV0007    DC    A(TESTLE)                             @01A     02-CEEEN
    CEEX0007    EQU *                                                   02-CEEEN
            ST    13,CEEDSABKC-CEEDSA(,1)  Set back chain.              02-CEEEN
            ST    0,CEEDSANAB-CEEDSA(,1)   Set new NAB value            02-CEEEN
            XC    CEEDSAFLAGS-CEEDSA(,1),CEEDSAFLAGS-CEEDSA(1) . Clear   02-CEEEN
            ST    1,CEEDSAFWC-CEEDSA(,13)  Set forward chain.           02-CEEEN
            LR    13,1                     Set save area address        02-CEEEN
            USING CEEDSA,13                Addresability to SF   V1R2M0  02-CEEEN
            MVC   CEEDSALWS,CEECAALWS-CEECAA(12) Get LWS addr   V1R2M0  02-CEEEN
            LR    1,2                                                   02-CEEEN
            BAL   1,*+8                                          @L2A 01-DFHEI
```

```
* The following gives an assembler message if DFHEISTG is too big  @P7A
        DS    0S((DFHEISTG+65264-DFHEIEND-4096)/4096)              @04C 01-DFHEI
        DC    AL2(DFHEIEND-DFHEISTG)  LENGTH OF STORAGE             @L2A 01-DFHEI
        DC    H'0'                 Parameter list version number    @P6C 01-DFHEI
***********************************************************************
*            ESTABLISH DATA ADDRESSIBILITY                       *
***********************************************************************
DFHEIPLR EQU  13                    PARAMETER LIST REGISTER      @BBAC81A 01-DFHEI
        LR    DFHEIPLR,15                                         @BBAC81A 01-DFHEI
        USING DFHEISTG,13                                         @BBAC81A 01-DFHEI
        MVC   DFHEIBP(L'DFHEIBP+L'DFHEICAP),0(1)                    @D3AX01-DFHEI
                                  COPY EIB AND CA PTRS         @D3A
***********************************************************************
*            ESTABLISH EIB ADDRESSIBILITY                        *
***********************************************************************
        L     DFHEIBR,DFHEIBP                                      @BBAC81A 01-DFHEI
        USING DFHEIBLK,DFHEIBR                                     @BBAC81A 01-DFHEI
***********************************************************************
*            END OF PROLOG CODE FOR EXEC INTERFACE               *
***********************************************************************
        MVC   OUTAREA(40),MSG1
        MVC   OUTAREA(4),EIBTRMID
*       EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
        DFHECALL =X'180660000800C20000082204000020',,(_____RF,OUTAREA*
              ),(FB_2,=Y(43))
***********************************************************************
        DS    0H                                                          01-DFHEC
        LA    1,DFHEIPL                                                    01-DFHEC
        LA    14,=X'180660000800C20000082204000020'                       01-DFHEC
        SR    15,15                                                       01-DFHEC
        LA    0,OUTAREA                                                    01-DFHEC
        STM   14,0,0(1)                                                    01-DFHEC
        LA    14,=Y(43)                                                    01-DFHEC
        ST    14,12(,1)                                                    01-DFHEC
        OI    12(1),X'80'          LAST ARGUMENT                           01-DFHEC
        L     15,=V(DFHEI1)                                                01-DFHEC
        BALR  14,15                INVOKE EXEC INTERFACE                    01-DFHEC
***********************************************************************
*       EXEC CICS RECEIVE
        DFHECALL =X'040200000800000014000040000000'
***********************************************************************
```

```
        DS    0H                                                01-DFHEC
        LA    1,DFHEIPL                                          01-DFHEC
        LA    14,=X'0402000008000000140000400000000'            01-DFHEC
        ST    14,0(,1)                                           01-DFHEC
        OI    0(1),X'80'         LAST ARGUMENT                   01-DFHEC
        L     15,=V(DFHEI1)                                      01-DFHEC
        BALR  14,15              INVOKE EXEC INTERFACE           01-DFHEC
***********************************************************************
        MVC   OUTAREA(13),MSG2
*       EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
        DFHECALL =X'180660000800C20000082204000020',,(_____RF,OUTAREA*
              ),(FB_2,=Y(13))
***********************************************************************
        DS    0H                                                01-DFHEC
        LA    1,DFHEIPL                                          01-DFHEC
        LA    14,=X'180660000800C20000082204000020'             01-DFHEC
        SR    15,15                                              01-DFHEC
        LA    0,OUTAREA                                          01-DFHEC
        STM   14,0,0(1)                                          01-DFHEC
        LA    14,=Y(13)                                          01-DFHEC
        ST    14,12(,1)                                          01-DFHEC
        OI    12(1),X'80'        LAST ARGUMENT                   01-DFHEC
        L     15,=V(DFHEI1)                                      01-DFHEC
        BALR  14,15              INVOKE EXEC INTERFACE           01-DFHEC
***********************************************************************
*       EXEC CICS RETURN
        DFHECALL =X'0E0800000800001000'
***********************************************************************
        DS    0H                                                01-DFHEC
        LA    1,DFHEIPL                                          01-DFHEC
        LA    14,=X'0E08000000800001000'                         01-DFHEC
        ST    14,0(,1)                                           01-DFHEC
        OI    0(1),X'80'         LAST ARGUMENT                   01-DFHEC
        L     15,=V(DFHEI1)                                      01-DFHEC
        BALR  14,15              INVOKE EXEC INTERFACE           01-DFHEC
***********************************************************************
*
MSG1    DC    C'xxxx: ASM program invoked. ENTER TO END.'
MSG2    DC    C'PROGRAM ENDED'
        DFHEIRET                 INSERTED BY TRANSLATOR
***********************************************************************
*            EPILOG CODE FOR EXEC INTERFACE                    *
***********************************************************************
        DS    0H                          @BBAC81A 01-DFHEI
        LA    1,CEET0014          Get address of termination list 02-CEETE
        L     15,=V(CEETREC)      Get address of termination rtn  02-CEETE
        BALR  14,15               Call termination routine.       02-CEETE

CEET0014 DC  A(*+8)             Parm 1                           02-CEETE
        DC    A(*+8+X'80000000')  Parm 2                          02-CEETE
        DC    A(0)               Enc_Modifier                     02-CEETE
        DC    A(0)               Return code.                     02-CEETE
CEEMAIN CSECT                                                     02-CEETE
CEEMAIN RMODE ANY                                                 02-CEETE
CEEMAIN AMODE ANY                                                 02-CEETE
        DC    A(TESTLE)                               @04A 02-CEETE
        DC    F'0'                                          02-CEETE
TESTLE  CSECT                                                     02-CEETE
***********************************************************************
*            END OF EPILOG CODE FOR EXEC INTERFACE             *
***********************************************************************
```

```
         LTORG ,                                             @BBAC81A 01-DFHEI
              =V(DFHEI1)
              =V(CEETREC)
              =Y(43)
              =Y(13)
              =X'180660000800C20000082204000020'
              =X'0402000008000000140000040000000'
              =X'0E0800000800001000'
         DS    0H                                            @F8E1S @L1C 01-DFHEI
         DFHEISTG               INSERTED BY TRANSLATOR
         DFHEIEND               INSERTED BY TRANSLATOR
*
***********************************************************************
*      P R O G R A M   P R O L O G   A R E A 1   ( P P A 1)          *
***********************************************************************
*
PPA10018   DS    0F                                            02-CEEPP
DFHPPA     DS    0F                                            02-CEEPP
           DC    AL1(PPANL0018-*) Offset to the entry name length 02-CEEPP
           DC    X'CE'           Language Environment Indicator.  02-CEEPP
           DC    B'10100000'                         . PPA flags  02-CEEPP
*                Bit 0   0 = Internal Procedure
*                        1 = External Procedure
*                Bit 1   0 = Primary Entry Point
*                        1 = Secondary Entry Point
*                Bit 2   0 = Block doesn't have a DSA
*                        1 = Block has a DSA
*                Bit 3   0 = compiled object
*                        1 = library object
*                Bit 4   0 = sampling interrupts to library
*                        1 = sampling interrupts to code
*                Bit 5   0 = not an exit DSA
*                        1 = Exit DSA
*                Bit 6   0 = own exception model
*                        1 = inherited (callers) exception model
*                Bit 7      Reserved
           DC    X'00'           Member flags                     02-CEEPP
           DC    A(PPA20018)     Addr of Compile Unit Block (PPA2) 02-CEEPP
           DC    A(0)                                             02-CEEPP
           DC    A(0)            Data Descriptors for this entry point 02-CEEPP
           DS    0H                                              02-CEEPP
PPANL0018 DC    AL2(6)      .  Length of Entry Point Name         02-CEEPP
           DC    CL6'TESTLE'  .  Entry Point Name                  02-CEEPP
CEEINPL   DS    0D                                              02-CEEPP
           DC    A(PPA2M0018)                                    02-CEEPP
           DC    A(CEEINPLSTST-CEEINPL)                          02-CEEPP
CEEINPLSTST DS 0F                                              02-CEEPP
           DC    X'00'           Control Level           @01A 02-CEEPP
           DC    X'00'           ENCLAVE=NO              @01A 02-CEEPP
           DC    X'00'                                   @01A 02-CEEPP
           DC    X'07'           Number of items.        @01C 02-CEEPP
           DC    A(PPA2M0018)  . A of A(first entry point in comp unit) 02-CEEPP
           DC    V(CEESTART)   . A(Address of CEESTART)            02-CEEPP
           DC    V(CEEBETBL)                                      02-CEEPP
           DC    A(15)         . Memeber id                        02-CEEPP
           DC    A(0)                                             02-CEEPP
           DC    XL4'00070000'          . EXECOPS(ON), PLIST      02-CEEPP
           DS    0H                                              02-CEEPP
*
```

```
   ***********************************************************************
   *         P R O G R A M   P R O L O G   A R E A 2   ( P P A 2)       *
   ***********************************************************************
   *
            EXTRN CEESTART                                        02-CEEPP
   PPA20018   DS    0F                                            02-CEEPP
            DC    AL1(15)        Member ID                        02-CEEPP
            DC    AL1(0)         Sub ID                           02-CEEPP
            DC    AL1(0)         Member defined                   02-CEEPP
            DC    AL1(1)         Level of PPAx control blocks      02-CEEPP
   PPA2S0018  DC    A(CEESTART)     A(CEESTART for this load module)  02-CEEPP
            DC    A(0)           A(Compile Debug Information (CDI) )  02-CEEPP
            DC    A(CEETIMES-PPA20018)   A(Offset to time stamp)    02-CEEPP
   PPA2M0018  DC    A(TESTLE)      . A(first entry point in comp. unit)  02-CEEPP
   *
   ***********************************************************************
   *                T I M E   S T A M P                                 *
   ***********************************************************************
   *
   *     Time Stamp
   *,Time Stamp = 2004/06/17 08:51:00                             02-CEEPP
   *,Version 1 Release 1 Modification 0                           02-CEEPP
   CEETIMES DS    0F                                              02-CEEPP
            DC    CL4'2004'      Year                             02-CEEPP
            DC    CL2'06'        Month                            02-CEEPP
            DC    CL2'17'        Day                              02-CEEPP
            DC    CL2'08'        Hours                            02-CEEPP
            DC    CL2'51'        Minutes                          02-CEEPP
            DC    CL2'00'        Seconds                          02-CEEPP
            DC    CL2'1'         Version                          02-CEEPP
            DC    CL2'1'         Release                          02-CEEPP
            DC    CL2'0'         Modification                     02-CEEPP
   ***********************************************************************
   * C O M M O N   A N C H O R   A R E A   ( C A A )             *
   ***********************************************************************
   ***********************************************************************
   ***********************************************************************
   LEPTRLEN EQU   4                                               03-CEEDN
   *
   CEECAA     DSECT ,              CAA mapping                     02-CEECA

      (Definition of LE CAA removed)

   *              TERMINATE DEFINITION OF DYNAMIC STORAGE            *
   DFHEISTG DSECT                                         @BBAC81A 01-DFHEI
            ORG                                                    01-DFHEI
   DFHEIEND DS    0X             END OF DYNAMIC STORAGE    @BBAC81A 01-DFHEI
            END
```

Changes to global user exits

Assembler programs translated with the LEASM option cannot be used as global user exit programs.

LEASM is used to produce Language Environment conforming MAIN programs in assembler.

Changes to task-related user exits

Assembler programs translated with the LEASM option cannot be used as task-related user exit programs.

LEASM is used to produce Language Environment conforming MAIN programs in assembler.

Part 4. Enterprise management

The CICSPlex SM element of CICS Transaction Server for z/OS, Version 3 Release 1 provides new capabilities that enable effective management of large runtime configurations by the use of modern user interfaces, so that you can meet your demanding service level objectives.

Chapter 11. CICSPlex SM Web User Interface enhancements

The CICSPlex SM Web User Interface has been improved to make it more powerful and more usable. The Web User Interface is now functionally equivalent to the CICSPlex SM TSO end user interface, and is now the primary method of accessing CICSPlex SM.

User favorites

All WUI users now have the ability to save tabular and detail views on an ad-hoc basis to an easily accessible and editable menu of favorites. This allows you to reach frequently used views with just one click. Administrators have the additional authority to update the favorites of other users.

For more information, see "User favorites" on page 286

User group profiles

Administrators are now able to create profiles for groups of users. These profiles contain information such as default context, scope, CMAS context, menu and result set warning count. In this way administrators can configure the WUI in different ways to suit different groups of users in order to present an interface that is more tailored to individual needs.

For more information, see "User group profiles" on page 290

Business application services redesign

The design of the business application services section of the WUI, used to manage CICS resource definitions, has been improved and simplified and should now be more familiar to users of CICS RDO.

For more information, see "Business application services redesign" on page 293

Record count warnings

The WUI can now be tailored to issue warnings before it opens a view that will generate large numbers of records. Following a warning you have the opportunity to alter the filters on the view in order to reduce the number of results returned. This improves WUI performance by reducing unnecessary waits.

For more information, see "Record count warnings" on page 296

Filter confirmation

The WUI view editor has been improved to enable you to include a filter confirmation panel before a view opens when you are creating or updating views. This means that when navigating to such a view, users will have the opportunity to amend filter content regardless of the size of result set that will be returned.

For more information, see "Filter confirmation" on page 297

Dynamic selection lists

The WUI now generates lists of potential values for various attributes in input panels. This enhances the usability of the interface by allowing you to select from a list of valid values, rather than having to remember them.

For more information, see "Dynamic selection lists" on page 298

Improved screen design

Several improvements have been made to maximize the use of screen space in WUI views and menus:

- The use of screen space on detail views has been improved by providing the ability to display the information in two columns rather than one. You can design your own two-column detail views using the view editor.
- The amount of white space on tabular views has been reduced by removing the **Select All** and **Deselect All** buttons, replacing them with icons that reside in the Record heading of the table.
- You can now collapse filters in order to provide more screen space for data. Collapsed filters can be expanded whenever necessary.

For more information, see "Improved screen design" on page 299

Requirements

There are no special hardware or software requirements to support this function.

Related information:

Chapter 27, "The CICS operating environment," on page 355

Changes to CICSPlex SM

User favorites

This topic describes the new user favorites feature of the CICSPlex SM Web User Interface.

User favorites give individual users of the WUI the ability to save selected WUI screens to a special menu where they can be accessed quickly and easily. A set of favorites always relates to an individual user only and not to user groups. However a similar result can be achieved for groups of users by using the new user group profile facility in conjunction with the view editor (see "User group profiles" on page 290 for guidance).

The information for user favorites is held in a new object called 'user'. The user object holds a menu group that contains hyperlinks for the favorites of a user. There are two ways of creating a user object:

- It can be created the first time a user creates a favorite. When a user creates a favorite by clicking on the icon, the WUI searches for a user object that has an ID matching the user's ID. If a matching user object cannot be found, the WUI server automatically creates a matching user object for that user.
- It can be created by an administrator in the user editor. See "Creating and managing favorites for other WUI users" on page 288 for more information.

Managing favorites with the favorites editor

This topic describes how you can use the new Web User Interface favorites editor to manage your own user favorites.

As well as being able to add views to your own favorites list using the icon, you can also work with your favorites using the favorites editor. To open this editor, click on **Favorites editor** in the **Special** section of the WUI navigation frame. This opens the editor in a new browser window as shown in Figure 31 on page 287.

Favorites Editor

Select an operation to perform on Favorites.

Operations

Move	Move an existing favorite
Copy	Copy an existing favorite
Edit	Edit an existing favorite
Delete	Delete an existing favorite

Special
New Home window
Close window
Sign off
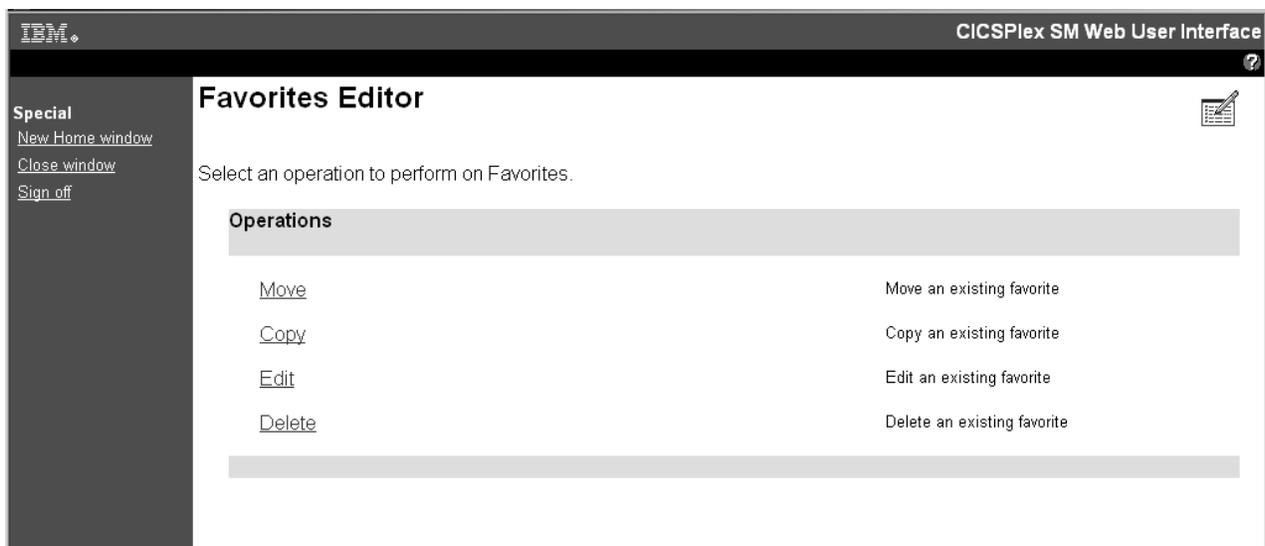
Figure 31. The favorites editor

To modify a favorite select a radio button and click on one of the four buttons at the bottom of the view.

1. To edit a screen, click **Edit**. Select the favorite you want to edit from the list, then click **OK** to open the **Components of Favorite** screen. From this screen you can choose to alter the menu's title and annotation, or destination. Select **Destination** to update its context and scope settings and filter settings.

2. To move the position of an item on the favorites list, click **Move**. Select one of your favorites from the list and click **OK**. This opens the **Move Favorite** screen. Choose the new position for the selected item and click **OK** again to make the change.

3. To copy a favorite screen, click **Copy**. Select one of your favorites from the list and click **OK**. This opens a **Copy Favorite** screen like the one shown in shown in Figure 32 on page 288.
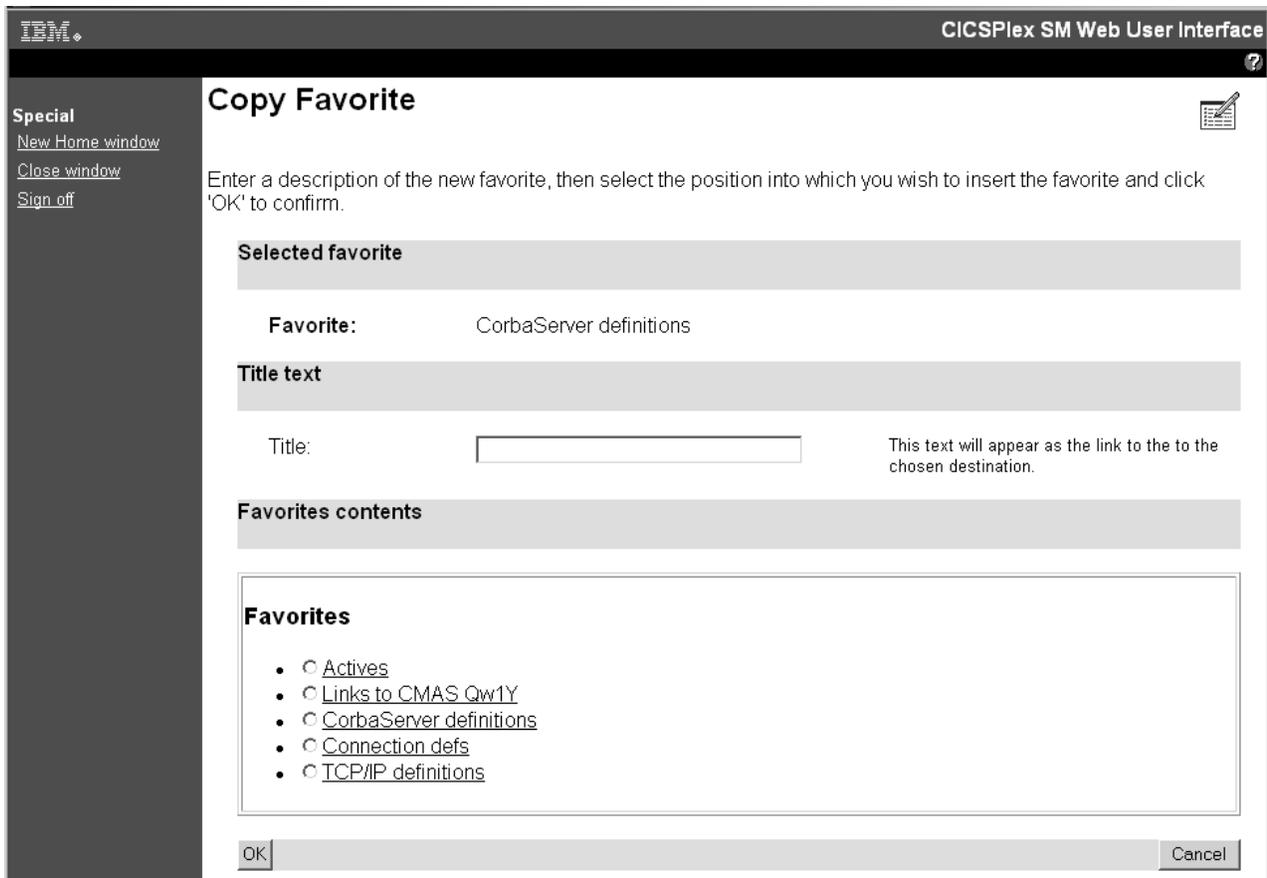
Figure 32. Copy Favorite screen

Now you can give the copy a new title. Select its position on your favorites list by selecting the existing favorite from the list above which the new copy will be positioned. If you want to position the new copy at the bottom of the list, do not select an existing favorite. Click **OK** to confirm the operation.

4. To delete a favorite, click **Delete**. Select one of your favorites from the list and click **OK** to confirm.

The favorites editor allows you to edit your own list of favorites. It does not allow you to manage the favorites of other users. Only WUI administrators with access to the user editor have the authority to do this.

Creating and managing favorites for other WUI users

This topic describes how administrators can use the new Web User Interface user editor to manage the favorites of other users.

In order to create and manage the favorites of other users you need to be a WUI administrator with at least update access to the new ESM facility profile EYUWUI.*wui_server_applid*.USER.

Figure 33 on page 289 illustrates screens in the user editor used to edit user favorites.
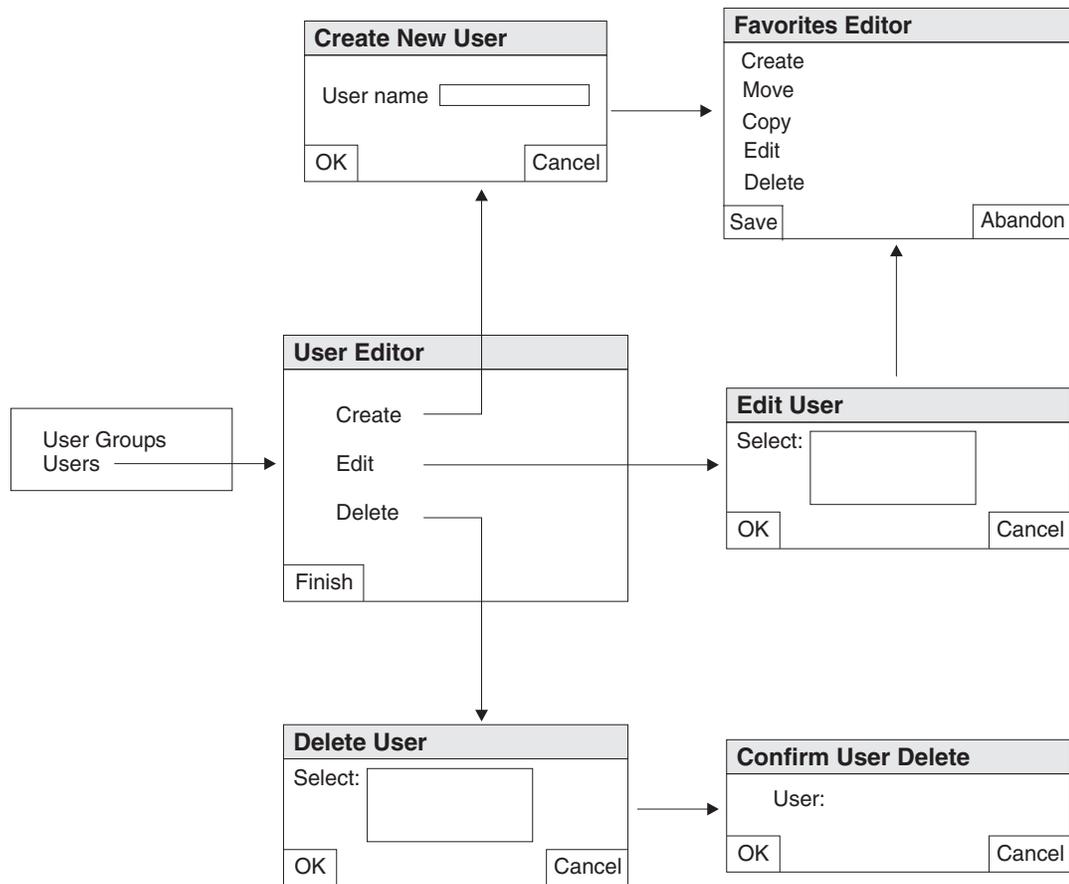
Figure 33. User screens in the user editor

If you have the necessary access, the hyperlink that launches the user editor appears in the navigation frame along with the view editor hyperlinks . Clicking on this hyperlink launches a new browser window displaying the **CICSPlex SM Web User Interface User Editor** screen. To work with favorites, select **Users**. This opens the **User Editor** screen. This screen gives you the following options:

Create
> Use this option to create a new user object in the repository. Each user needs a user object to hold a list of favorites. The WUI automatically creates a user object the first time a user creates a favorite (if one does not already exist), so most existing users will already have one.

Edit Use this option to work with the favorites of an existing user. It allows you to create, move, copy, edit and delete a user's favorites.

Delete Use this option to delete a user object from the repository. This also deletes any existing favorites for that user.

This is how you would use the user editor to create a new user object and add some favorites:

1. Create the new user object
   a. Starting at the **User Editor** screen, click **Create** to open the **Create New User** screen.

b. Type in the new user ID. User IDs are restricted to a maximum of eight characters. Valid characters are `A-Z` , `0-9`, `#` , `$` and `@`.

c. Click **OK**. This confirms the operation and opens the **Favorites Editor** screen used to create and manage a user's list of favorites.

2. Create a new favorite item.

a. In the **Favorites Editor** screen, click **Create**. This opens the edit screen that is used by the view editor for editing detail items. From this screen you can perform most of the actions on a favorite that you can on a menu item in the view editor.

b. Complete this screen by giving the new favorite a title and type in the target view or menu, the context and scope settings, and the filter settings.

c. Click **Save**. This saves the updates in the repository and returns you to the **User Editor** screen. If you click **Abandon**, all of your changes, including the new user object, are discarded.

3. Create more favorites and add them to the new user object.

a. From the **User Editor** screen, click **Edit** This opens the **Edit User** screen

b. Select the newly created user ID and click **OK**. This takes you back to the **Favorite Editor** screen.

c. Create a new favorite item as before, and save your updates. Repeat this step for each favorite you want to add to the new user object.

Related tasks:

"Managing favorites with the favorites editor" on page 286
This topic describes how you can use the new Web User Interface favorites editor to manage your own user favorites.

User group profiles

This topic describes the new user group profiles feature of the CICSPlex SM Web User Interface (WUI).

This new feature enables administrators to create profiles for groups of WUI users. What individual users see and do when they log on to the WUI can be controlled by the user group profile to which they belong enabling the WUI to be tailored to the needs of various groups of users. You can, for example, use the view editor to create a new WUI menu containing only operations views and make this the default menu for a group of users. This would provide a simplified operational WUI for users who only need to carry out this kind of task.

A user group profile can contain the following information:

• Result set warning count (see "Record count warnings" on page 296)

• Name of the default main menu

• Name of the default navigation menu

• Default context

• Default scope

• Default CMAS context

If a profile does not have some of these values specified; for example, if the default menu value is blank, the corresponding WUI parameter value specified in the JCL is used. This is also the case for any invalid values set in the profile, for example if the specified default menu does not exist.

User group profiles are created and managed using a new editing facility, the user editor, which can be accessed by administrators with the necessary authority from the WUI main menu. See "Creating and managing user group profiles" for more information.

To facilitate this a new type of object called a user group has been introduced. User group objects, like view and menu objects, can be imported and exported using the COVC transaction.

User group profiles can be used only if security is active in the WUI. A user group object relates to a user group name in the external security manager (ESM). When a user signs on to the WUI the signon procedure gets the accessor environment element (ACEE) from the ESM for the user. The ACEE is used to retrieve the user's default group. The default group name is then used to obtain a corresponding group object in the WUI. If there is no corresponding group object found in the WUI cache when the user signs on, the default values specified in the WUI parameters in the JCL are used for that user.

Note: In the ESM user groups can contain only the following characters:
  - A through Z (you can enter lowercase characters but they are folded to uppercase)
  - 0 through 9
  - # (X'23'), $ (X'24') and @ (X'40')

In order to manage user group profiles an administrator needs update access to a new ESM facility profile named `EYUWUI.`*`wui_server_applid`*`.USER` where *wui_server_applid* is the application ID of the WUI server to which this profile relates.

Setting up a new user group involves the following steps:
1. Creating a new user group profile in the WUI using the user editor (see "Creating and managing user group profiles" for guidance.
2. Creating a user group with the same name in the ESM.
3. Setting the user group as one or more users' default user group in the ESM.

Creating and managing user group profiles

This topic describes how to create and manage Web User Interface (WUI) user group profiles.

User group profiles are created and managed using a new WUI editing facility called the user editor.

The hyperlink that launches the user editor appears in the navigation frame along with the view editor hyperlinks. A signed on user can see a user editor hyperlink only if he or she has at least update access to a the new ESM facility profile `EYUWUI.`*`wui_server_applid`*`.USER`. Clicking on this hyperlink launches a new browser window that uses a set of new views created for the user editor.

When you launch the user editor you can choose to edit user groups or users. The diagram in Figure 34 on page 292 illustrates the relationships between the screens in the user group section of the user editor.
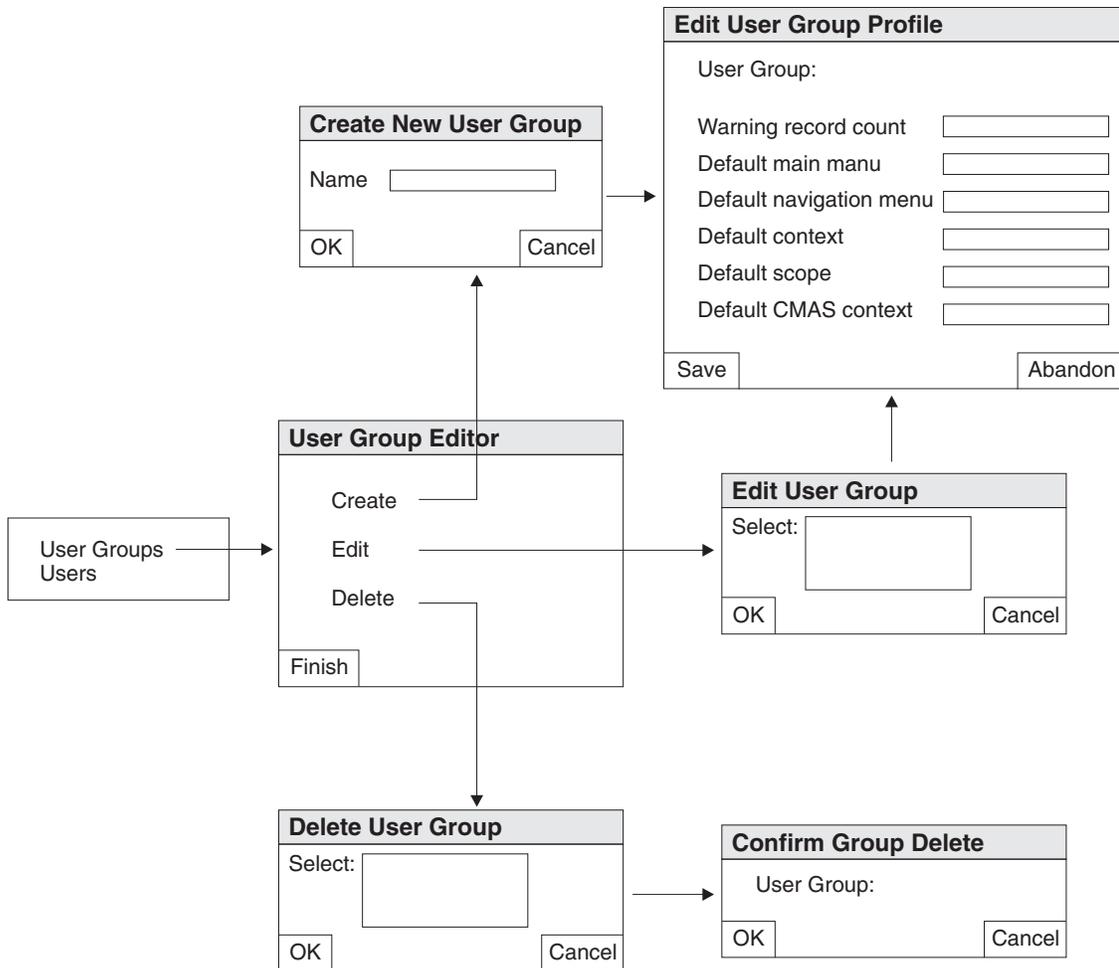
Figure 34. User group screens in the user editor

Using the user editor is straightforward. For example to create a new profile follow this procedure:

1. Navigate to the **Create New User Group** screen:
   a. Click **User Editor** in the navigation frame to open a new **CICSPlex SM Web User Interface User editor** window.
   b. Click **User Groups** to open the **User Group Editor** screen. This screen gives you options to create, edit or delete a profile.
   c. Click **Create** to open the **Create New User Group** screen.

2. Name the new group profile:
   a. Type in the name of the new group. (This will need to match a user group name defined in the ESM.) Names are restricted to a maximum of eight characters. As in the ESM, the allowed characters are `A-Z` (upper case) `0-9`, `#`, `$` and `@`
   b. Click **OK**. This confirms the operation and opens the **Edit User Group Profile** screen.

3. Type in the user group details. See "User group profiles" on page 290 for a description of this information.

4. Click **Save** to create the new profile or **Abandon** to cancel the operation.

To edit a group, click on **Edit** in the **User Group Editor** screen to open the **Edit User Group** screen and select an existing group. This opens an **Edit User Group Profile** screen containing details of the selected group. Make your changes and click **Save** to update the profile or **Abandon** to cancel the operation.

To delete a group, click on **Delete** in the **User Group Editor** screen to open the **Delete User Group** screen and select an existing group. Click **OK** to open the **Confirm User Group Delete** screen and **OK** again to confirm the operation.

Business application services redesign

This topic describes improvements to the design of the business application services views and menus used for CICS resource definitions in the WUI.

In the new WUI design BAS functions are separated into basic and fully functional view menus. To access BAS functions from the WUI main menu click **Administration views**. At the bottom of the Administration views are two new sub-menus:

Basic CICS resource administration views
> This provides a simplified RDO-like model of BAS including resource definitions, resource groups and resource descriptions but not resource assignments.

Fully functional Business Application Services (BAS) administration views
> In addition to the basic model this also includes links to resource assignment views aimed at more advanced users. This adds more power and flexibility to the management of resource definitions.
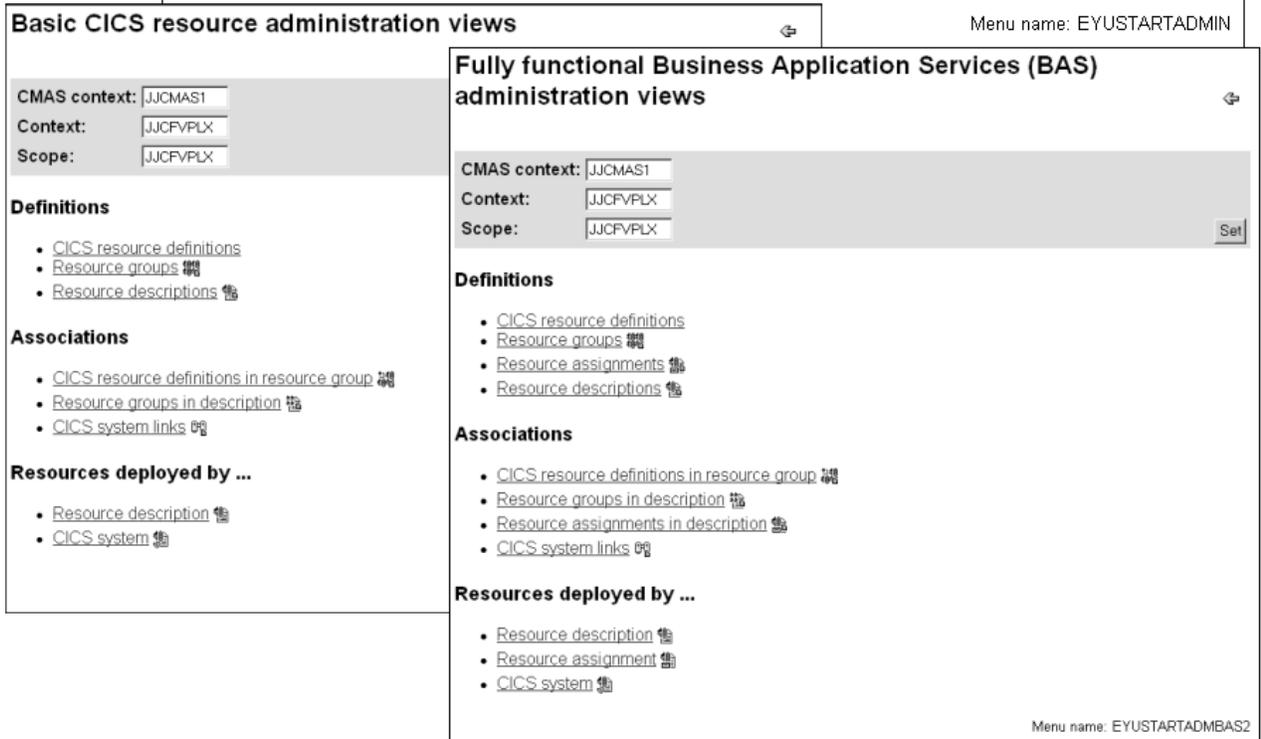
Figure 35. Detail of new CICS resource definition BAS menus

Links from both menus are split into three groups:

Definitions
Includes the following links:
CICS resource definitions
Menu containing links to definition views for each resource type.
Resource groups
Link to definitional view for managing resource group definitions. Associated actions are Create, Update, Remove, Install and Add to Resource description.
Resource assignments (fully functional menu only)
Link to definitional view for creating and managing resource assignments. Associated actions are Create, Update, Remove and Add to Resource description.
Resource descriptions
Link to definitional view for creating and managing resource descriptions. Associated actions are Create, Update, Remove, Install and Replace.

Associations
Includes the following links:
CICS resource definitions in resource group
Link to a new view as shown in Figure 36 on page 295, which is supported by the new RESINGRP resource table. This replaces the **Resources in resource group** set of views (one for each resource

type) of previous releases.
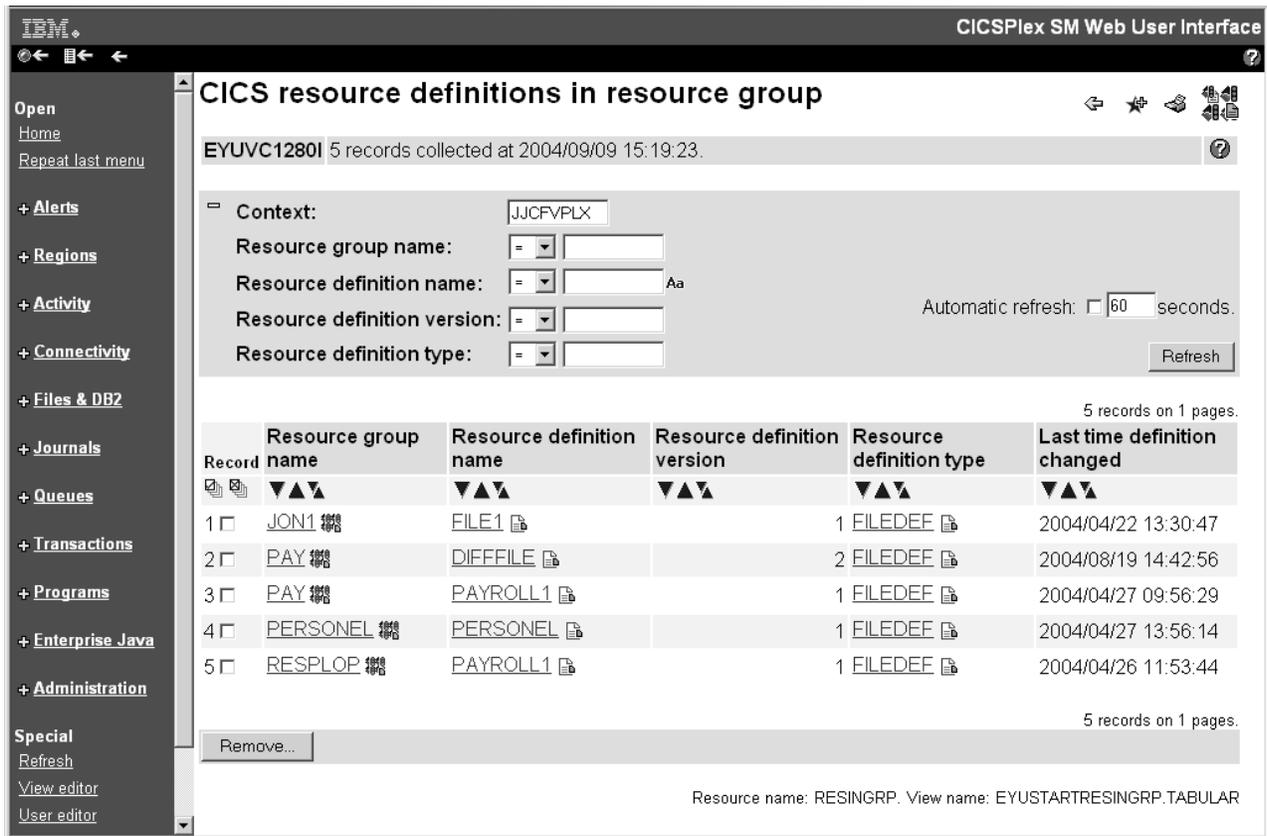
Figure 36. CICS resource definitions in resource group tabular view

The view includes a **Remove** action button allowing you to remove an association between a resource definition and its parent resource group. There is no create action with this view. Adding a resource to a group is carried out while defining the resource itself

Resource groups in description
Link to definitional view for managing the associations between resource groups and resource descriptions. Associated actions are Create, Update and Remove.

Resource assignment in description (fully functional menu only)
Link to definitional view for managing the associations between resource assignments and resource descriptions. Associated actions are Create, Update and Remove.

CICS system links
Link to definitional view for managing CICS system link definitions. Associated actions are Create, Remove and Install.

Resources deployed by...
Includes the following links to views displaying active CICS resources:

Resource description
Link to a tabular view displaying deployed resources selected by resource description.

Resource assignment (fully functional menu only)
Link to a tabular view displaying deployed resources selected by resource assignment.

CICS system
> Link to a tabular view displaying deployed resources selected by CICS system.

Record count warnings

This topic describes how the WUI can issue warnings if the expected number of records to be returned in certain views exceeds a set threshold.

This new feature is designed to issue a warning if a request to open certain views will result in a larger than expected amount of data. This provides an opportunity to alter the filters on these views and confirm or cancel the request before the request is executed.

This feature applies to the views associated with the following resources:

CMDT	FEPINODE	PROGRAM
CONNECT	FEPIPOOL	REMFILE
DB2CONN	FEPITRGT	REMTDQ
DB2ENTRY	INDTDQ	REMTRAN
DB2TRN	INTRATDQ	RQMODEL
DOCTEMP	JOURNAL	SMFJRNL
DSKJRNL	JRNLNAME	SYSDUMP
DSNAME	LOCFILE	TAPEJRNL
ENQMODEL	LOCTRAN	TCPIPS
EXITGLUE	MODENAME	TERMNL
EXITTRUE	PARTNER	TRAN
EXTRATDQ	PROCTYP	TRANDUMP
FEPICONN	PROFILE	TSMODEL

There are two ways of specifying the number of records required to trigger the warning mechanism:

- Set a value for the new DEFAULTWARNCNT WUI server initialization parameter during WUI server configuration. DEFAULTWARNCNT can take an integer value in the range of 0 to 99999999. This is an optional parameter.
- Set a warning record count value in a user group profile. This allows WUI administrators to set a value for all the members of a WUI user group. See "Creating and managing user group profiles" on page 291 for more information.

In both cases the default value is 0 meaning that no warnings are issued.

A value set in a user group takes precedence over a value set in the DEFAULTWARNCNT parameter. Before a warning is issued the WUI checks to determine whether or not the signed on user is associated with a user group (this can occur only if the WUI is running with security switched on). If so, the WUI uses any value for the warning count specified in that user group. If the user does not belong to a user group, or no maximum value is set, the WUI checks the initialization parameters and uses any value set there. If no value has been specified, the default of 0 is used.

If the value for the warning count is greater than 0, the WUI checks the size of the potential set of records to be returned. This gives an indication of the maximum

number of records that could be returned. It is not always accurate because any filters other than that associated with the first part of the primary key field are not taken into account. If this potential value is greater than the value of the warning count, a warning screen like the one in Figure 37 is displayed. If the returned value is less than or equal to the warning count value, the WUI view is displayed in the usual way.
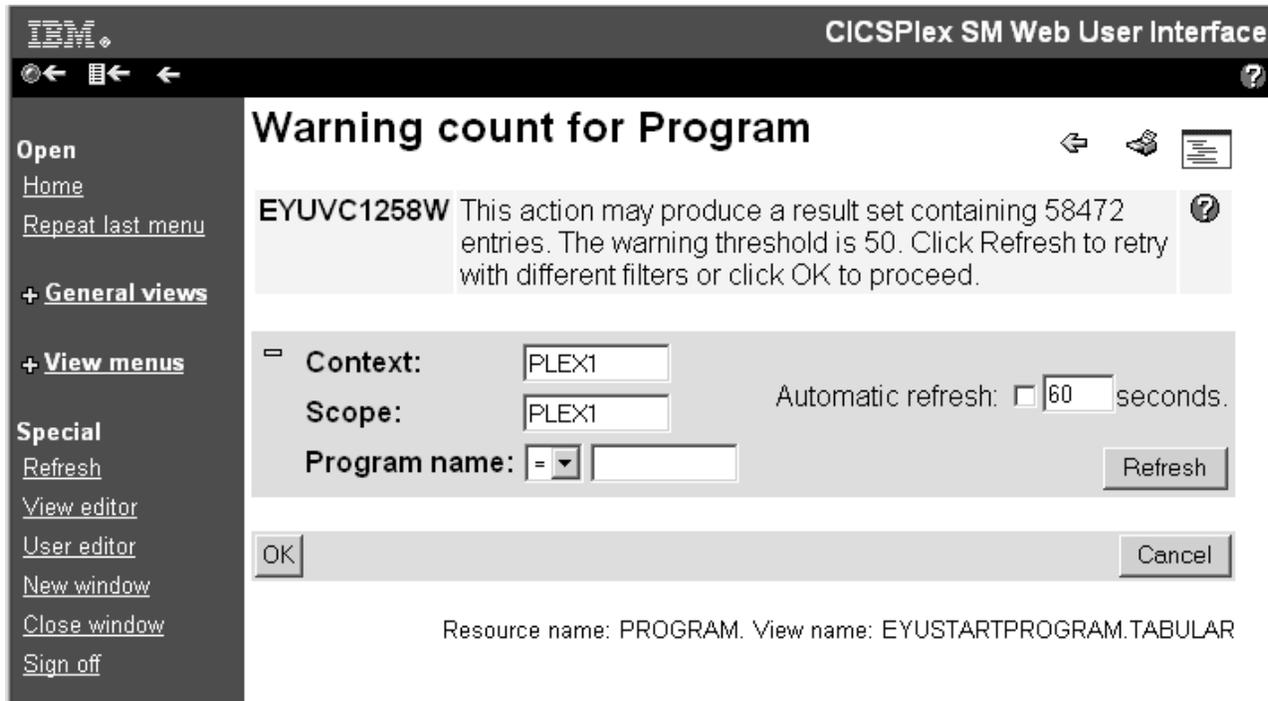
Figure 37. Warning count screen

Clicking **Refresh** drives a new request. This allows you to alter the primary key field, which is included in the **Warning count screen**, to test a new filter value in order to produce a smaller set of results. Any other filters remain in place from the original request. If the set of results returned is not below the warning count limit, the warning count screen is redisplayed with the original warning count message. If the size is below the warning count limit, the warning count screen is redisplayed.

Clicking **OK** at the bottom of the screen performs the current request regardless of the number of results to be returned.

Clicking **Cancel** is the equivalent of pressing the back link on the browser; the previous screen is redisplayed.

Filter confirmation

This topic describes the new filter confirmation feature of the CICSPlex SM Web User Interface.

It is now possible for you to specify filter criteria before the WUI retrieves the data for a view in a similar way to the TSO EUI.

When designing your own views in the view editor you now have the option to specify that a filter confirmation screen should be displayed before the data retrieval is executed. The views supplied with the WUI are not affected by this change. A

typical filter confirmation screen is shown in Figure 38.
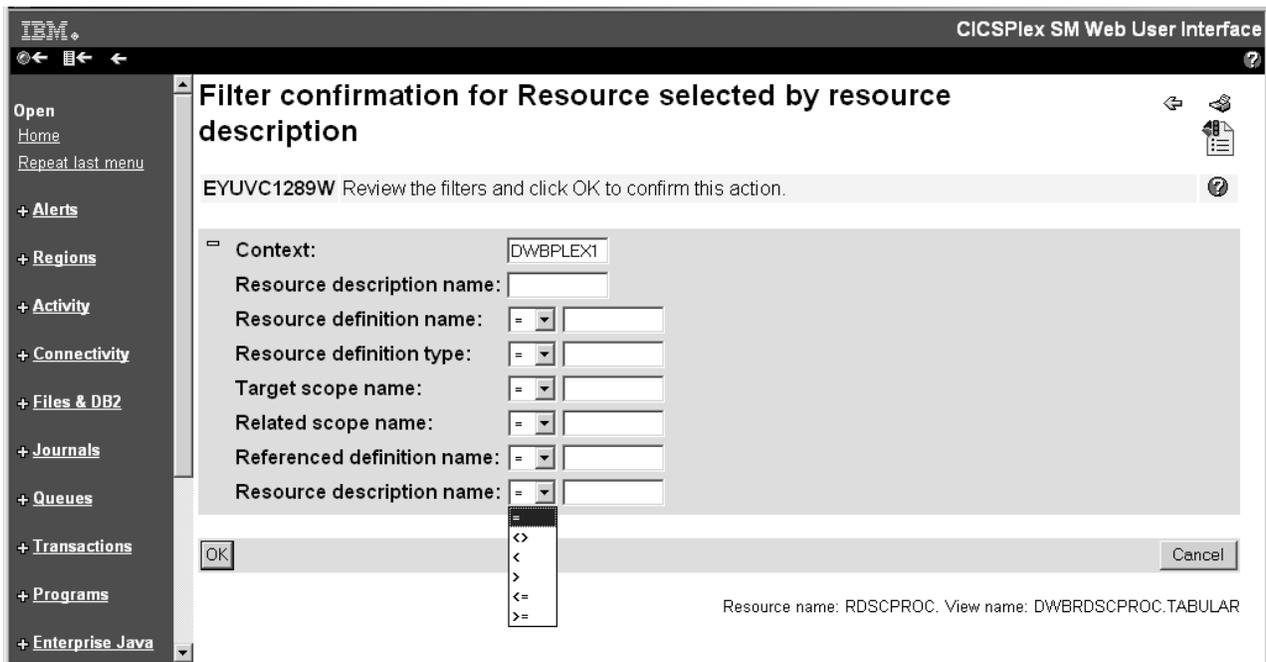
Figure 38. Filter confirmation screen

You can override the filter values in the confirmation screen by overtyping the fields and clicking **OK**.

Note: Filter confirmation is always applied before any result set warnings are processed.

Dynamic selection lists

This topic describes the new dynamic selection lists feature of the CICSPlex SM Web User Interface (WUI).

The WUI is now capable of generating dynamic selection lists in a similar way to the TSO EUI. Fields capable of generating a selection list, are marked with the icon

to the right of the text input box. For resource input fields, clicking the icon opens a screen like this:
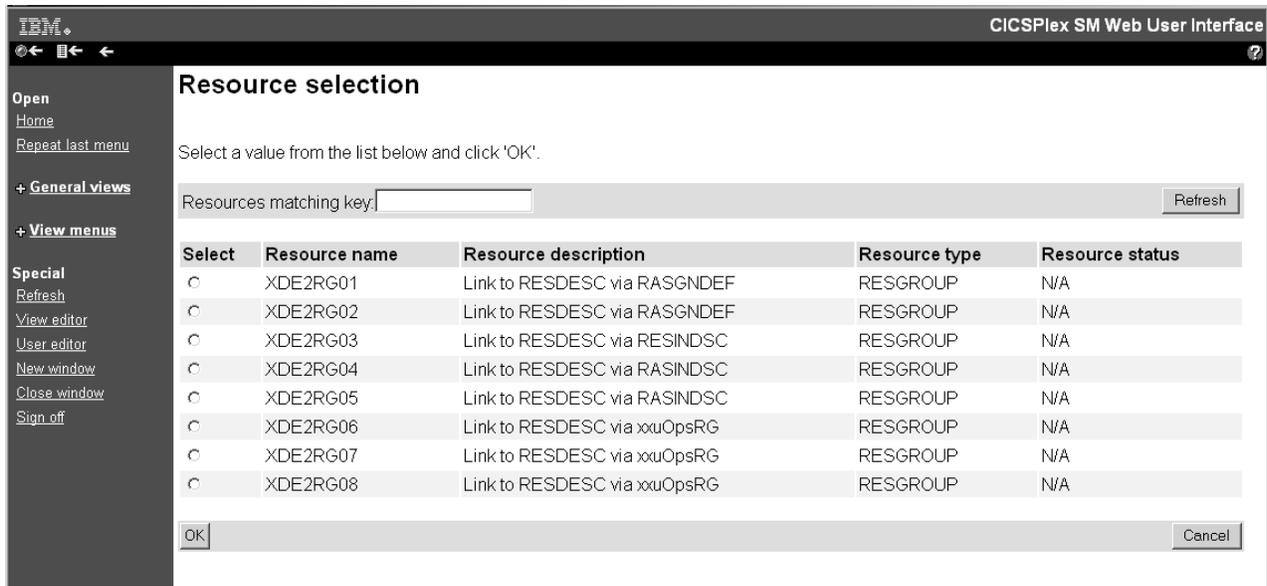
Figure 39. Resource selection list screen

The valid values are displayed on the screen in tabular form sorted by resource type. To make a selection from the list you just select a radio button and click **OK**.

Selection list screens include an entry field containing the value currently being used to filter the list. To use a new filter, type a new value in the entry field and click **Refresh**.

Only attributes and parameters in entry fields in task guides, tabular and detail views are capable of generating selection lists. Filters and context and scope fields are not able to generate selection lists.

The selection list screen for metadata attributes requests show all, or a filtered subset, of the attributes for a base table sorted by attribute name. They are used mainly for evaluation definition creation.

You can reduce the number of values displayed in a selection list by specifying filter values. These filters differ from filters as they normally occur in the WUI because they use only the = (equals) operator. You can type a generic value as the value for the attribute using the wildcard symbols *(asterisk) and + (plus sign) in the usual way, for example, `DW*`. On a selection list screen you can then alter the values displayed in the list by modifying the filter value that appears in the box at the top of the screen.

Note: If you type in a value that does not contain generic characters, this value is still used as the filter regardless of whether any values will be displayed in the selection list. This differs from the current EUI behavior, which would produce a list of all available values if no values are produced when it is supplied with an attribute that does not contain wild cards.

Improved screen design

This topic describes new features that improve the usability and appearance of WUI views and menus.

Two-column detailed views

The introduction of two-column detailed views increases the amount of data that can be displayed on your screen and reduces the need for screen scrolling. Here is an example of a two-column view:

IBM® CICSPlex SM Web User Interface

Terminal

EYUVC1280I 2 records collected at 09/07/04 13:52:23.

CMAS context: IYK3ZDB1
Context: DWBPLEX1
Scope: DWBPLEX1   Refresh

Open
Home
Repeat last menu

+ Alerts
+ Regions
+ Activity
+ Connectivity
+ Files & DB2
+ Journals
+ Queues
+ Transactions
+ Programs
+ Enterprise Java
+ Administration

Special
Refresh
View editor
User editor
New window
Close window
Sign off

CICS system name	IYK3ZDB3	Terminal ID	CBRF
Access method	Vtam	Acquire status	Notapplic
Alternate page height	0	Alternate page width	0
Alternate printer		Hardware COPY feature for alternate printer	Notapplic
Alternate screen height	0	Alternate screen width	0
Alternate-map-set suffix	'00'X	APL keyboard feature	Noaplkybd
APL text feature	Noapltext	ASCII data stream type	Notapplic
Automatic transaction initiation (ATI) status	Ati	Audible alarm feature	Audalarm
Session binding status	Notapplic	Background transparency feature	Backtrans
Extended color feature	Color	Console ID	
Copy feature in control unit	Nocopy	Correlation ID	
Session creation status	Notapplic	Device data stream type	Ds3270
Default page height	24	Default page width	80
Default screen height	24	Default screen width	80
Device type	T3277r	Device busy status	Not applicable
Disconnect Requests status	Discreq	Dual-case keyboard status	Nodualcase
Exit tracing status	Notapplic	Extended data stream support	Extendedds
Function management header (FMH) option	Nofmhparm	Forms feed feature	Noformfeed
Graphic character set global ID	0	Code page global ID	0
Horizontal form feature	Nohform	Extended highlight feature	Hilight
Number of input messages	0	Katakana terminal	Nokatakana
Selector pen feature	Nolightpen	Real link connection for remote TOR	
Last map referenced in SEND MAP command		Last map set referenced in SEND MAP command	
Mode name		Magnetic slot reader	Msrcontrol
National language ID	E	Terminal definition type	Model
Network name	CBRF	Next transaction ID	
Network qualified name		Outboard formatting support	Noobformat
Outboard operator IDs used	Noooperid	Operator ID	

Figure 40. Part of a two-column detail view

You can create two-column detailed views using the view editor. See "Creating two-column detailed views" on page 302 for guidance.

Expand and collapse filters

Filters on tabular views can take up large amounts of screen space restricting the data that can be displayed. You can now mitigate this by collapsing the filters when you are not altering them. You can expand them again when you want to change them. Figure 41 shows part of a tabular view with the filters expanded.

Figure 41. Expanded filter area of a tabular view

To collapse the filter, click the collapse (minus sign) icon at the top left of the filter area.

Figure 42 shows the same screen with the filters collapsed. The amount of information displayed is the same but the data fields run across the screen.
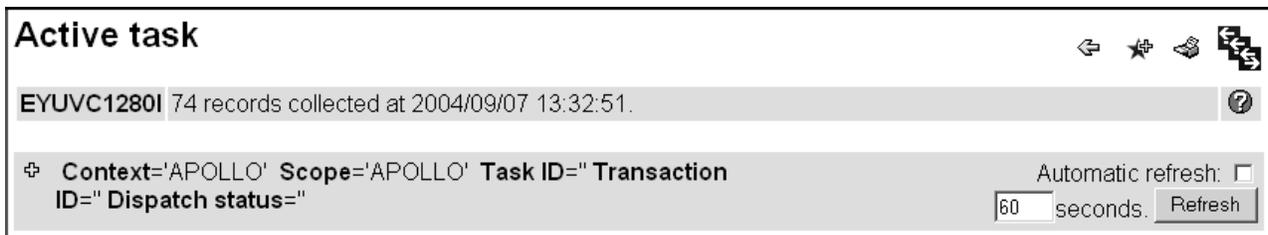
Figure 42. Collapsed filter area

To expand the filters, click the expand filter (plus sign) icon.

In order to determine whether filters should be displayed in their expanded or collapsed state by default use this new optional WUI server initialization parameter:

FILTERSTYLE(EXPAND/COLLAPSE)

If you do not specify this parameter, the WUI displays filters in the expanded state by default.

When you navigate away from a view with filters in a non-default state and view some other screen, the filter area in the new view reverts to the default filter setting as specified in the FILTERSTYLE parameter.

If you navigate away from a tabular view and then immediately return, the filter area remains in the state you left it in when the view was last displayed.

Select all and deselect all icons

In order to improve the utilization of space on tabular views, the **Select all** and **Deselect all** buttons used in previous releases to select or deselect all resources matching given criteria have been replaced by the following icons:

 Select all

 Deselect all

Both icons are located in tabular views in the second row of the column labeled **Record** (in the same table row as the sort and summary icons).

The functionality of the select all and deselect all options is otherwise unchanged.

Creating two-column detailed views

This topic explains how to create and update two-column detailed views using the WUI view editor.

The WUI view editor has been updated to enable the creation and update of two-column detailed views. The **Add View** screen of the view editor has been updated to include a new view type called **Two column detail form**. The **Detail form** type of previous releases has been renamed **One column detail form**.

The process of creating a new view is much the same as in previous releases. This is an outline of the steps you need to follow to add a new two-column detailed view to an existing view set:

1. Open the view editor and navigate to the **Add View** screen.

   a. From the Main menu navigation panel, click **View editor** > **View sets** > **Edit**. This opens the **Open View Set** screen.

   b. Select the view set to which your new detailed view will belong and click **OK**.

      Note: Remember, it is not possible to change an IBM-supplied view set or view unless you copy and rename it first.

      This opens the **View Set Contents** screen, which contains a list of the views currently belonging to the selected view set.

   c. Click **Add** at the bottom of the screen. This opens the **Add View** screen, which is used to create a new view for the selected view set.

2. Name the new view and define its type.

   a. Type in a name for your new view in the **View name** field.

   b. Select **Two column detail form** from the list of view types.

   c. Select **Key attributes** from the list of pre-fill options and click **OK**. This opens the **Detailed Form Components** screen.

3. Add an attribute to the left column

   When you choose to add an item to a two-column detailed view, the new element is placed in the left column. In order to maintain left-right alignment a white space element is automatically placed in the right column opposite the new item.

   a. Select **Form contents**. This opens a **Form Contents** screen similar to the one in Figure 43 on page 303. Notice it contains a Space element type, which is necessary in the creation of two column screens in order to balance

the right and left columns.
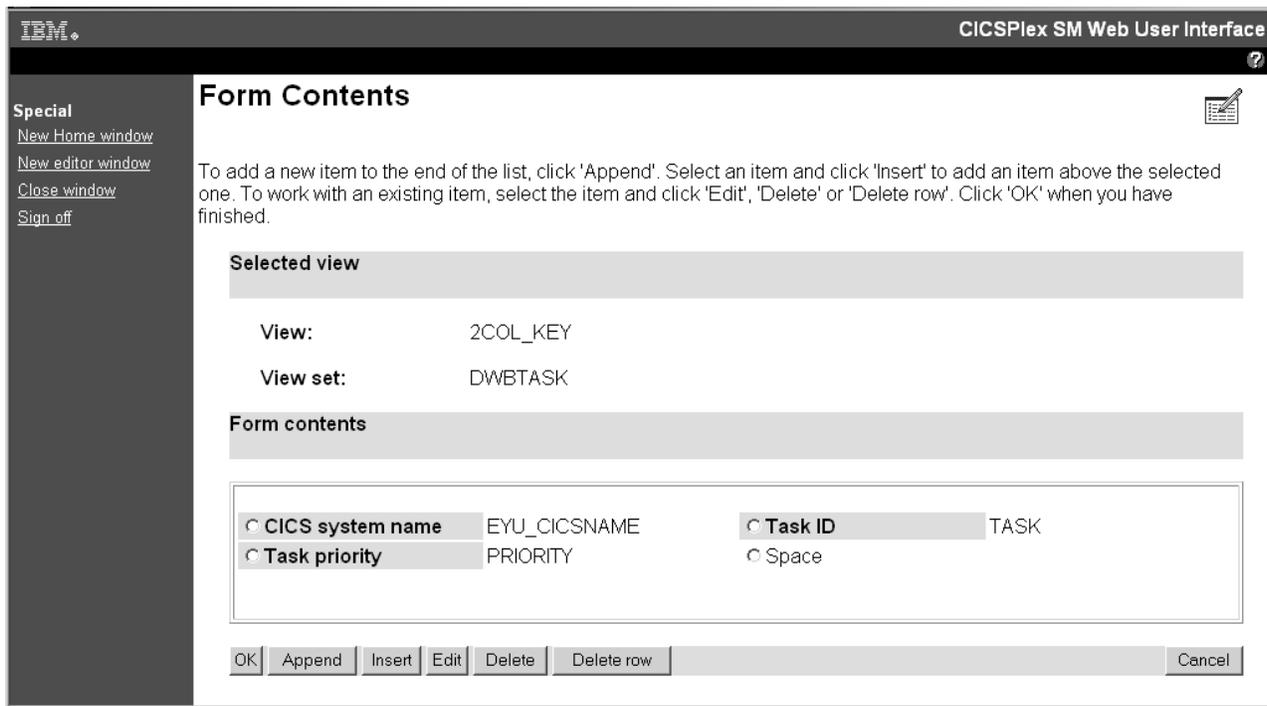
Figure 43. Form Contents screen for a two-column detailed view

> b. Click **Append**. This opens the **Form Item Type** screen.
>
> c. Select **Attribute field** and click **OK** to open the **Form Attribute Field** screen.
>
> d. Select an attribute from the list and click **OK** to open the **Form Item Components** screen.
>
> e. Complete the definition of the new attribute by:
>   - Typing in the attribute title and any annotation
>   - Selecting the display options; either normal or graphical
>   - Adding any view links.
>
> f. Click **Finish** to add the new attribute to your new view. This takes you back to the **Form Contents** screen.

4. Add another attribute to the right hand column of the view.

   In order to add an element to the right hand column, you need to select a white space element and then edit it to change it to the type of element required:

   > a. Select the white space element created in step 3 and click **Edit**. This opens the **Form Item Type** screen again.
   >
   > b. Select the type of element you want to add to the right hand column and follow the procedure outlined in step 3 to define it.
   >
   >   Click **Finish** to add the right hand element to the view

   You can repeat steps 3 and 4 as many times as necessary to add more elements to the left and right columns.

   If you want to remove an item from the right hand column, select it and click **Delete**. This converts the item back to a white space element.

   Note: You cannot delete an individual white space element but you can remove adjacent elements in both columns by clicking **Delete row**.

5. Complete the view definition.

   a. Click **OK** on the **Form Contents** screen to return to the **Detailed Form Components** screen.

   b. Now add the rest of the components of the new view including a title, action buttons, filters, context and scope and so on.

   c. When you have added all the required components, click **OK** to save the new view and return to the **View Set Contents** screen.

Changes to CICSPlex SM API

Some changes have been made to the CICSPlex SM API to support the new business application services design.

API changes in support of business application services redesign

A new resource table named RESINGRP is introduced. The RESINGRP resource describes the membership of resource definitions in a resource group. It has the following attributes:

Table 12. RESINGRP resource table attributes

Name	Data type	Source	Length	Sum	Description
CHANGETIME	DATETIME	CPSM	8	MAX	Last time the definition was changed
RESGROUP-1	CHAR	CPSM	8	DIF	Resource group name
RESNAME-2	CHAR	CPSM	8	DIF	Resource definition name
RESVER-3	BINARY	CPSM	1	AVG	Resource definition version
RESTYPE-4	CHAR	CPSM	8	DIF	Resource definition type
DESCRIPTION	CHAR	CPSM	58	DIF	Resource definition description

Messages

A number of new messages have been added to the CICSPlex SM Web User Interface.

New Web User Interface client messages

EYUVC1258W	EYUVC1289W	EYUVC1293I	EYUVC1294I
EYUVC1310I	EYUVC1311I	EYUVC1312W	EYUVC1313E
EYUVC1314I	EYUVC1321E		

New Web User Interface editor messages

EYUVE0111E	EYUVE0142I	EYUVE0144I	EYUVE0760I
EYUVE0761I	EYUVE0821I	EYUVE0949I	EYUVE0950E
EYUVE0951E	EYUVE0952E	EYUVE0953E	EYUVE0954E

EYUVE0956I	EYUVE0957E	EYUVE0958E	EYUVE0959E
EYUVE0960E	EYUVE0961I	EYUVE0962I	EYUVE0963E
EYUVE0964E	EYUVE0965E	EYUVE0969I	EYUVE0970E
EYUVE0971E	EYUVE0972E	EYUVE0973E	EYUVE0974E
EYUVE0975E	EYUVE0976E	EYUVE0977E	EYUVE0978I
EYUVE0979I	EYUVE0980E	EYUVE0981E	EYUVE0982E
EYUVE0983I	EYUVE0984E	EYUVE0985E	

Chapter 12. Enhancements to CICSPlex SM batched repository update facility

New methods have been introduced to provide alternatives to the existing TSO end-user interface BATCHREP command for submitting batched updates to a specified CICSPlex SM repository.

The alternative ways of accessing BATCHREP are as follows:
- A new BATCHREP resource table available as an object for reference by the CPSM API
- WUI support of the new BATCHREP resource table
- A batch utility program that provides a BATCHREP facility

The WUI support and the batch utility exploit the use of the new BATCHREP resource table as an object referenced by the CPSM API.

Benefits of the enhancements to CICSPlex SM batched repository-update facility

The introduction of these alternatives to the use of the CICSPlex SM TSO end-user interface provides:
- Improved accessibility of CPSM BATCHREP facilities through a new BATCHREP view in the WUI
- Extension of access to CPSM BATCHREP facilities to a new z/OS job step utility and to a CPSM API program
- Provision of a batch utility enabling definitions to be maintained from a job step and thereby allowing maintenance to be integrated into library control systems

Requirements

There are no special hardware or software requirements to support this function.

Related information:

Chapter 27, "The CICS operating environment," on page 355

Batch utility program

The new batch utility program runs as a z/OS job step and connects to the CMAS whose repository is to be updated. Batched repository updates are submitted to run in that CMAS. The utility can initiate EXECUTE and CHECK actions. The CMAS must be at CICS Transaction Server for z/OS Version 3.1.

Changes to CICSPlex SM application programming interface

The BATCHREP resource table can now be used with the CICSPlex SM API.

Two actions have been added to the BATCHREP resource table:

EXECUTE

Submits batched repository updates to run in a specified CMAS. The CONTEXT option on the CONNECT command of the CICSPlex SM API is used to specify the name of the CMAS.

CHECK

> Checks the commands specified in the batched repository-update facility input file.

The input parameter fields for these actions match the external attributes as shown in Table 13:

Table 13. BATCHREP resource table attributes

Name	Datatype	Source	Length	Summary	Description
(INPUTDSN-1)	CHAR	CPSM	44	DIF	Input dataset name
(INPUTMEMBER-2)	CHAR	CPSM	8	DIF	Input member name
PRINTCLASS	CHAR	CPSM	1	DIF	Output print class
PRINTNODE	CHAR	CPSM	8	DIF	Destination print node
OUTPUTUSER	CHAR	CPSM	8	DIF	Output user ID

Changes to the CICSPlex SM Web User Interface

A new WUI view called Batch Repository Update Job (EYUSTARTBATCHREP) has been added to the Administration Views menu (EYUSTARTADMIN). To access the new view from this menu click **Batched repository update job** under the Administration views heading. The new view is equivalent to the TSO EUI BATCHREP view.

To submit batched repository updates using the WUI you must:

- Ensure that the WUI server is connected to the CMAS that is associated with the data repository you want to update.
- Select a record and click the **Check** or **Execute** button
- Enter the input parameters that are equivalent to those required by the TSO EUI BATCHREP view.

Messages

A number of new data repository messages have been added:

New data repository messages

EYUXD0901E	EYUXD0902E	EYUXD0903E	EYUXD0904E
EYUXD0905E	EYUXD0906E	EYUXD0907E	EYUXD0908I
EYUXD0909I	EYUXD0910E	EYUXD0911E	EYUXD0912E
EYUXD0913E	EYUXD0914E	EYUXD0915E	EYUXD0916E

Security

Existing CICSPlex SM security features that support the use of an external security manager can still be used to implement access controls on the use of the new z/OS job step utility, the API programs that reference the BATCHREP resource table, and the use of the WUI to reference the BATCHREP view.

Migration

The new WUI or CPSM API access to BATCHREP, or the z/OS job step utility for BATCHREP are facilities to use as alternatives to the existing TSO EUI BATCHREP facility.

The CMAS to which the utility connects must be at CICSPlex SM Version 3.1

Part 5. Miscellaneous changes

This part describes new and changed function in CICS Transaction Server for z/OS, Version 3 Release 1 that are outside the scope of the three themes for the release.

Chapter 13. New installation process

This topic draws your attention to a new installation process for CICS.

This release of CICS Transaction Server is installed using the SMP/E RECEIVE, APPLY, and ACCEPT commands. The SMP/E dialogs may be used to accomplish the SMP/E installation steps.

The process is described in the *CICS TS 3.1 Program Directory*. It is in line with IBM Corporate Standards, and may be familiar to those who have installed other z/OS products.

The traditional method, DFHISTAR, of installing CICS Transaction Server is still available. The *Program Directory* indicates where information about DFHISTAR may be found in the *Installation Guide*.

Benefits of the new installation process

The SMP/E RECEIVE, APPLY, and ACCEPT process for installation is an IBM Corporate Standard. It may be familiar to those who have installed other z/OS products.

Chapter 14. EXTRACT STATISTICS command

A new SPI command, `EXTRACT STATISTICS`, handles statistics for URIMAP, PIPELINE, and WEBSERVICE resources.

Use the `EXTRACT STATISTICS` command to retrieve the current statistics for a single resource, or global statistics for a class of resources.

The `EXTRACT STATISTICS` command performs a function equivalent to `COLLECT STATISTICS`, for the URIMAP, PIPELINE, and WEBSERVICE resources. To collect statistics for other resources use the existing `COLLECT STATISTICS` command.

The syntax of the `EXTRACT STATISTICS` differs from that of `COLLECT STATISTICS`.

Benefits of the EXTRACT STATISTICS command

All CICS SPI commands are restricted in the number of distinct options they can support. As new resources have been added to CICS over time, the limit has been reached for the `COLLECT STATISTICS` command, and it is not possible to accommodate the new URIMAP, PIPELINE, and WEBSERVICE resources on the existing command.

The `EXTRACT STATISTICS` command uses the RESTYPE option, with a CVDA, to specify a CICS resource. As a result, there is no limit on the number of resources that the command can potentially support, although in this release, only the three new resources are supported

Changes to CICS externals

Changes to the system programming interface
The `EXTRACT STATISTICS` command

The `EXTRACT STATISTICS` is added to CICS to supplement the function of the `COLLECT STATISTICS` command.

For the PIPELINE, URIMAP, and WEBSERVICE resources, use `EXTRACT STATISTICS` to retrieve the current statistics for a single resource, or global statistics for a class of resources. `EXTRACT STATISTICS` only deals with these three resources. To COLLECT STATISTICS for other resources, continue to use the `COLLECT STATISTICS` command.

EXTRACT STATISTICS performs a function equivalent to COLLECT STATISTICS for the resources URIMAP, PIPELINE, and WEBSERVICE, because these could not be provided by extending COLLECT STATISTICS (due to a design limitation of that command). When compared to `COLLECT STATISTICS`, the syntax of EXTRACT STATISTICS is different and provides for unlimited future expansion.

Extract STATISTICS

```
►►──EXTRACT STATISTICS──RESTYPE(cvda)─────────────────────────────────────────►
                                      └─RESID(data-area)──────────────────┐
                                                         └─RESIDLEN(data-value)─┘

►─SET(ptr-ref)──────────────────────────────────────────────────────────────►◄
              └─LASTRESET(data-area)──────────────────────────────────────┐
              └─LASTRESETHRS(data-area)──LASTRESETMIN(data-area)──LASTRESETSEC(data-area)─┘
```
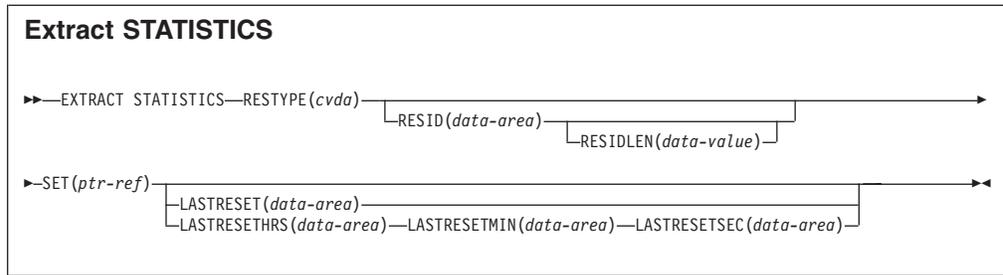
Conditions: INVREQ, IOERR, LENGERR, NOTAUTH, NOTFND

Description

The EXTRACT STATISTICS command returns to the invoking application the current statistics for a particular resource, or overall statistics for the resources of a given type.

The statistics that CICS gives you are those that have been accumulated after the expiry of the last statistics extraction interval, end-of-day expiry, or requested reset. (Statistics already written to the SMF data set cannot be accessed.) The EXTRACT STATISTICS command does not cause the statistics counters to be reset.

CICS obtains enough storage for the data returned from this command, and returns a pointer to this area. The first two bytes of the area contain its length. This storage can be reused by subsequent EXTRACT STATISTICS commands, so you should store elsewhere any data that is required beyond the next issue of the command. CICS releases this storage at task termination.

Not all resource types provide both global and specific statistics. Table 14 tells you which statistics are available for each resource type, and gives the copybook name for each set of available statistics. The copybooks define the format of the returned statistics. Where no copybook name is given in the global statistics column, global statistics are not available for the resource type; similarly, where there is no entry in the specific statistics column, you cannot get statistics for an individual resource.

Table 14. Resource types and statistics

Resource type	CVDA	RESIDLEN	Statistic type	Global statistics	Specific statistics
PIPELINE	1124	Char(8)	PIPELINE	--	DFHPIRDS
URIMAP	1173	Char(8)	URIMAP	DFHWBGDS	DFHWBRDS
WEBSERVICE	1174	Char(32)	WEBSERVICE	--	DFHPIWDS

Copybooks are provided in ASSEMBLER, COBOL, and PL/I. (There is no copybook for C.) The names of the copybooks are the same in each language. You can find them in the following libraries:

ASSEMBLER	CICSTS31.CICS.SDFHMAC
COBOL	CICSTS31.CICS.SDFHCOB
PL/I	CICSTS31.CICS.SDFHPL1

Note: Some of the copybooks contain packed fields. Before these fields are used, they should be checked for hexadecimal zeros. The COBOL versions of the fields have been redefined as numeric with a suffix of -R for this purpose.

Options

LASTRESET(*data-area***)**
> returns a 4-byte packed decimal field giving the time at which the counters for the requested statistics were last reset. This is usually the time of the expiry of the last interval. The last reset time is always returned in local time.

> There are two formats for the reset time:
> - A composite (packed decimal format 0hhmmss+), which you obtain by using the LASTRESET option.
> - Separate hours, minutes, and seconds, which you obtain by specifying the LASTRESETHRS, LASTRESETMIN, and LASTRESETSEC options respectively.

LASTRESETHRS(*data-area***)**
> returns a fullword binary field giving the hours component of the time at which the counters for the requested statistics were last reset (see the LASTRESET option).

LASTRESETMIN(*data-area***)**
> returns a fullword binary field giving the minutes component of the time at which the counters for the requested statistics were last reset (see the LASTRESET option).

LASTRESETSEC(*data-area***)**
> returns a fullword binary field giving the seconds component of the time at which the counters for the requested statistics were last reset (see the LASTRESET option).

RESID(*data-area***)**
> specifies the name of the resource for which statistics are being extracted. The absence of this keyword means that global statistics are to be extracted.

RESIDLEN(*data-value***)**
> specifies the length of the RESID data area. If omitted, the default value is the length given in Table 14 on page 316.

RESTYPE(*cvda***)**
> requests statistics for a particular resource type depending on the CVDA supplied. Valid CVDA values are:

> **PIPELINE**
> > requests statistics for a PIPELINE; RESID identifies the particular PIPELINE.

> **URIMAP**
> > requests statistics for a URIMAP; RESID identifies the particular URIMAP.

> **WEBSERVICE**
> > requests statistics for a WEBSERVICE; RESID identifies the particular WEBSERVICE.

SET(*ptr-ref***)**
> specifies a pointer reference to be set to the address of the data area containing the returned statistics. The first 2 bytes of the data area contain the length of the data area in halfword binary form.

Chapter 15. Support for mixed case passwords

When the security manager used with CICS supports the use of mixed case passwords, CICS Transaction Server for z/OS, Version 3 Release 1 does not convert passwords to uppercase before passing them to the security manager.

There are several places where you can enter a password in CICS:

Resource definitions

> The following resource definitions have a PASSWORD attribute:
>> FILE
>> TCPIPSERVICE
>> TERMINAL

API commands

> The following commands have a PASSWORD option:
>> CHANGE PASSWORD
>> VERIFY PASSWORD
>> SIGNON

The signon transaction (CESN)

> The transaction offers two fields where passwords may be entered:
>> Password
>> New password

In all these cases, the way CICS handles these passwords depends upon whether the external security manager used with CICS supports mixed case passwords, or not:

- If the security manager supports mixed case passwords, then CICS passes the password you specify to the security manager unchanged.
- If not, then CICS converts the password to uppercase before passing it to the security manager.

Chapter 16. Codepage conversion changes

CICS is enhanced to provide a means of converting between EBCDIC or ASCII and Unicode data. Conversion can occur in either direction.

- These conversions make use of the z/OS Unicode conversion services. Typically such conversions take place as part of the processing of HTTP requests or the new CONTAINER API commands introduced in CICS TS 3.1.
- CICS already supports the conversion of data between any of the range of EBCDIC and ASCII codepage combinations listed in *CICS Family: Communicating from CICS on System/390*.
- This support is extended to permit the conversion of data between any of them and either UTF-8 or UTF-16. Conversion between the UTF-8 and UTF-16 forms of Unicode is also supported.

To use the data conversion methods described here, as opposed to those offered by earlier releases of CICS, you must be communicating your data using channels or containers, as described in the *CICS Application Programming Guide*.

CICS documents and document templates cannot be converted to or from the UTF-8 and UTF-16 character encodings. This restriction applies whether they are used as a static response in CICS Web support, retrieved by CICS in response to `EXEC CICS WEB API` commands, or retrieved by an application program using an `EXEC CICS DOCUMENT RETRIEVE` command.

Appendix F of the *z/OS Support for Unicode: Using Conversion Services* manual -SA22 -7649 records those conversions which are supported though these services. These are not limited to Unicode, but include the ability to convert between a broad range of character encodings, including EBCDIC, ASCII and Unicode.

Note: The conversion between 037 and 500, as used, for example, with the MQ transport is an EBCDIC to EBCDIC conversion brought about by small differences in the character encodings used by CICS and MQ.

CICS now supports any of these character conversions by making use of the z/OS conversion services. However, those conversions that earlier releases of CICS carried out using a set of tables, continue to be supported in that manner. It is only if CICS TS 3.1 is asked to carry out a conversion between a pair of CCSIDs that are unsupported via these tables, that it attempts the conversion using the z/OS services.

Benefits of Codepage conversion changes

CICS is now able to:

- accept UTF-8 and UTF-16 data as inputs
- convert received data to another encoding format, and
- subsequently convert results back to the appropriate UTF form on return.

The range of codepage pairs available for conversions is considerably extended.

Data for conversion is expected to come primarily from HTML, XHTML, and XML, but the new function is not limited to these, input data, from any interface that observes conventions for identifying itself as UTF, is processed with suitable conversion.

Terminology

The following terms have been added to the CICS Glossary.

CCSID
See "coded character set identifier".

coded character set identifier
A 16-bit number identifying a specific set of encoding scheme identifier, character set identifier(s), code page identifier(s), and additional coding-related required information, that uniquely identifies the coded graphic character representation used. Acronym: CCSID.

UNICODE
A universal character encoding standard that supports the interchange, processing, and display of text that is written in any of the languages of the modern world. It also supports many classical and historical texts in a number of languages. The Unicode standard has a 16-bit international character set defined by ISO 10646.

Requirements

The hardware and software requirements for these enhancements to Codepage conversion are the same as for CICS TS generally.

CICS implements these enhancements by making use of the z/OS Unicode conversion services, provided by z/OS.

Changes to CICS externals

Changes to installation

No changes are needed as part of the CICS installation process. However, if you have requirements which depend on support for the conversion of UTF-8 or UTF-16 data, you must enable the z/OS conversion services and install a conversion image which specifies the conversions that you want CICS to perform.

Refer to the instructions in the *z/OS Support for Unicode: Using Conversion Services* manual SA22-7649 to find out the steps needed to set up and configure conversions supported though the operating system services.

If z/OS conversion services are not enabled, a message is issued by CICS to indicate this. That message can be suppressed if you do not need these services.

- If the message is encountered when starting a CICS region that is expected to make use of these services, then an IPL is necessary to enable the z/OS conversion services.

To discover the status of z/OS conversion services after an IPL, use one of these commands from an MVS console:

/D UNI To show whether z/OS conversion services were enabled.

/D UNI,ALL
To show whether z/OS conversion services were enabled, and which conversions are supported by the system.

For details of this, see the *z/OS Support for Unicode: Using Conversion Services* manual SA22-7649

Changes to system initialization parameters

There is a new system initialization for the enhancement to Data Conversion. The new parameter is:

`LOCALCCSID={037|CCSID}`
> Specifies the default CCSID for the local region.
>
> The CCSID is a value of up to 8 characters. If CCSID value is not specified, the default `LOCALCCSID` is set to 037. For lists of valid CCSIDs, .see
> * *CICS Family: Communicating from CICS on System/390*, and
> * Appendix F of the *z/OS Support for Unicode: Using Conversion Services* manual -SA22 -7649 .

Changes to application programming

The following information is added to the *CICS Application Programming Guide*.

Data conversions within CICS using z/OS Unicode conversion services adds to the data conversion support that is provided for any of the range of EBCDIC and ASCII codepage combinations listed in *CICS Family: Communicating from CICS on System/390*.

Conversions involving Unicode typically take place as part of the processing of HTTP requests or the use of the CONTAINER API commands introduced in CICS TS 3.1. The conversion of data to or from either UTF-8 or UTF-16 and EBCDIC and ASCII codepages, depends on the selection of suitable conversion images. Conversion between the UTF-8 and UTF-16 forms of Unicode is also supported.

To use the data conversion methods described here, as opposed to those offered by earlier releases of CICS, you must be communicating your data using channels or containers, as described in the *CICS Application Programming Guide*.

CICS documents and document templates cannot be converted to or from the UTF-8 and UTF-16 character encodings. This restriction applies whether they are used as a static response in CICS Web support, retrieved by CICS in response to EXEC CICS WEB API commands, or retrieved by an application program using an `EXEC CICS DOCUMENT RETRIEVE` command.

Appendix F of the *z/OS Support for Unicode: Using Conversion Services* manual -SA22 -7649 records those conversions which are supported though these services. CICS now supports any of these character conversions by making use of the z/OS conversion services.

Ensuring that required conversion images are available
> Those CCSIDs used as part of CICS applications must be made known to the System Programmers responsible for maintaining the z/OS Conversion Image, so that specific conversions are available to the CICS regions where these applications execute.

Handling CCSID 1200

> CICS supports conversions involving UTF-16 data using any of the following CCSID's: 1200, 1201, and 1202. The z/OS conversion services permit CCSID 1200, in its big-endian form, to be used, but does not contain support for the little-endian form or for CCSIDs 1201 or 1202. CICS transforms any source data that is identified in any of these unsupported forms to the big-endian form of 1200 before passing the data to z/OS for conversion. If the target data is one of the unsupported forms then CICS

receives the data as the big-endian form of 1200 and transforms it to the required CCSID. If the target CCSID is 1200 then CICS assumes the encoding to be in big-endian form. If the conversion is between any of these CCSIDs then CICS will carry out the transformation without calling the z/OS conversion services.

When setting up the z/OS conversion image for conversions involving any of these forms of UTF-16 then CCSID 1200 must be specified. CCSIDs 1201 and 1202 will not be recognised by z/OS when attempting to create a conversion image.

CICS respects the byte order marker for inbound conversions, but is not able to retain that information when handling a related outbound conversion. All outbound data for CCSID 1200 is UTF16-BE. Application programmers need to know about this and perform their own BE to LE conversions if they so require.

Sharing a conversion image
- Unless the PTF for APAR OA05744 is applied, do not specify a search order for those conversions, installed into the z/OS image which are intended for use by CICS.
- If the same conversions are needed for COBOL you must define the conversion image with two separate statements:
    - one with no search order, and
    - the other explicitly specifying a search order of 'RECLM'.

    for example:.

    ```
    CONVERSION 850,037;
    CONVERSION 850,037,RECLM;
    ```

    With the APAR installed, CICS and COBOL can make use of those supported conversions which specify the default search order implicitly or explicitly, removing the need to provide two control statements in the image generation file.

JAVA programs
Codepage conversion facilities exist within JAVA, So it is not neccessary to duplicate these in CICS. The conversion facilities described here do not extend to JAVA programs. For an example, see *Java applications in CICS*.

Changes to CICS utilities
Dump formatters

There is a new component within the AP domain which shows the new control blocks. The component identifier is CV.

Changes to problem determination

Messages

Two new messages are issued resulting from failures in invoking the z/OS conversion services.
- A console message is issued during CICS initialisation to indicate that Unicode conversion is not supported by this CICS region because the services are not enabled.

- A message is issued to report that a particular conversion between two specific CCSIDs is not supported by this system.

Abends
- No new abend codes are introduced.
- Disaster responses are percolated back to the caller of the new function, by the recovery routines in DFHCCNVG.

Chapter 17. Simplified definition of default code pages

The default client or server code pages can be defined in the system initialization table in order to reduce the number of conversion tables required to configure a CICSplex.

Whenever data is passed between CICS and another system, some or all of the data may have to be converted from ASCII to EBCDIC format, or vice versa. Data conversion is facilitated by the DFHCNV conversion table, which contains a conversion template for each resource for which conversion is required.

Certain distributed components of a CICSplex such as CICS Transaction Gateway for z/OS and CICS Transaction Server for Windows do not provide an override for the default client code page specified in the conversion table. Because conversion tables do not have a suffix, two tables can be required, each residing on a different library and differing only in the default code page.

In order to reduce the number of conversion tables required, you can now specify that the default client or server code page is defined in the system initialization table.

For the client code page:
1. In the DFHCNV TYPE=ENTRY and TYPE=SELECT macros, specify the value SYSDEF for the CLINTCP parameter.
2. In the system initialization table, set a default client code page by specifying a value for the CLINTCP parameter. You can use any value supported for the CLINTCP parameter on the DFHCNV macro. The default is CLINTCP=437.

For the server code page:
1. In the DFHCNV TYPE=ENTRY and TYPE=SELECT macros, specify the value SYSDEF for the SRVERCP parameter.
2. In the system initialization table, set a default server code page by specifying a value for the SRVERCP parameter. You can use any value supported for the SRVERCP parameter on the DFHCNV macro. The default is SRVERCP=037.

Benefits of improved defaults for code pages in data conversion templates

The ability to define code page defaults in the system initialization table instead of directly in DFHCNV macros can simplify the definition and management of a CICSplex by reducing the number of DFHCNV conversion tables that have to be maintained.

Requirements

There are no special hardware or software requirements.

Changes to CICS externals

Changes to system initialization parameters

In order to define default client and server code pages in the system initialization table, two new system initialization parameters are introduced: CLINTCP and SRVERCP, matching the existing DFHCNV macro parameters of the same name.

New system initialization parameters

CLINTCP={437│codepage}
> Specifies the default client code page to be used by the DFHCNV data conversion table but is used only if the CLINTCP parameter in the DFHCNV macro is set to SYSDEF. The *codepage* is a field of up to 8 characters and can take the values supported by the CLINTCP parameter in the DFHCNV macro.

SRVERCP={037│codepage}
> Specifies the default server code page to be used by the DFHCNV data conversion table but only if the SRVERCP parameter in the DFHCNV macro is set to SYSDEF. The *codepage* is a field of up to 8 characters and can take the values supported by the SRVERCP parameter in the DFHCNV macro.

Changes to user-replaceable programs
New DFHCNV macro parameter operand

The new operand SYSDEF has been added to the TYPE=INITIAL and TYPE=ENTRY macro parameters CLINTCP and SRVERCP. These macros define the user-replaceable data conversion table DFHCNV.

- The DFHCNV TYPE=INITIAL macro defines the beginning of the conversion table. It gives a list of valid code pages.
- The DFHCNV TYPE=ENTRY macro specifies a name and type to uniquely identify a data resource. There must be one for each resource for which conversion is required.

The format of the changed parameters is now as follows:

CLINTCP={437│SYSDEF│nnnn [, nnnn, ...]}
> The first operand defines the default client code page to be used when the CLINTCP and CDEPAGE operands are omitted from a DFHCNV TYPE=ENTRY macro.
>
> SYSDEF specifies that the default client code page is determined by the system initialization table parameter CLINTCP.
>
> You can specify further code pages; they are validated but are not used.

SRVERCP={037│SYSDEF│nnnn [, nnnn, ...]}
> The first operand defines the default client code page to be used when the SRVERCP and CDEPAGE operands are omitted from a DFHCNV TYPE=ENTRY macro.
>
> SYSDEF specifies that the default client code page is determined by the system initialization table parameter SRVERCP.
>
> You can specify further code pages; they are validated but are not used.

Chapter 18. 64-Bit Addressing Toleration changes

CICS can now provide meaningful information when those tasks, which make use of 64-bit addressing architecture, abend.

CICS does not support 64-bit addressing execution, but programs can use storage at addresses which are only available when CICS is running on 64-bit architecture machines. These changes provide an extension to the CICS abend capture mechanisms so that the contents of the full 64-bit general purpose registers is captured.

Benefits of 64-Bit Addressing Toleration changes

Customers developing 64-bit addressing code, now have access to more information when a task abends.

Requirements

The hardware and software requirements for these enhancements are the same as those for the rest of this release of CICS TS.

Changes to CICS externals

Changes to CICS utilities

Dump formatters

The CICS dump formatter displays the contents of the 64-bit General Purpose Registers captured when the abend occurred.

Changes to problem determination

The contents of the full 64-bit general purpose register file are captured and made available to you.

There are no new Messages, Abend codes, or trace points.

Chapter 19. Support for revoked user IDs

\# When the command **EXEC CICS VERIFY PASSWORD** is issued, CICS now enforces the
\# revoked status of a user ID or a user's group connection. For example, if a user has
\# tried to log on too many times, the id is revoked and the user cannot access the
\# system or resources.

Part 6. Discontinued function

Some functions which were supported in CICS Transaction Server for z/OS, Version 2 have been discontinued, or reduced in scope in CICS Transaction Server for z/OS, Version 3 Release 1.

Chapter 20. Withdrawal of runtime support for OS/VS COBOL programs

Run-time support for OS/VS COBOL programs is withdrawn.

OS/VS COBOL programs, which had runtime support in CICS Transaction Server for z/OS, Version 2, cannot run under CICS TS for z/OS, Version 3.

OS/VS COBOL programs must be upgraded to Language Environment conforming COBOL, and recompiled against a level of COBOL compiler supported by CICS. Enterprise COBOL for z/OS and OS/390 Version 3 is the recommended compiler.

Chapter 29, "High-level language support," on page 363 and the *CICS Application Programming Guide* for CICS TS for z/OS, Version 3 have information about supported compilers for COBOL and other languages. Appendix B of the *CICS Application Programming Guide* provides assistance with converting OS/VS COBOL programs to Language Environment conforming COBOL.

A new abend code ALIK indicates an attempt to use an OS/VS COBOL program. In this situation, CICS abnormally terminates the task and disables the program, and CICS processing continues.

Chapter 21. Changes to BTAM and TCAM support

CICS support for the Basic Telecommunications Access Method (BTAM) is discontinued in CICS Transaction Server for z/OS, Version 3 Release 1. Support for the Telecommunications Access Method (TCAM) is limited to indirect support for the DCB interface.

Withdrawal of BTAM support

CICS Transaction Server for z/OS, Version 3 Release 1 does not support the Basic Telecommunication Access Method (BTAM).

For several CICS releases, BTAM terminals have been supported only indirectly: that is, by transaction routing from a back-level terminal-owning region (TOR) to which the terminals were attached. In CICS TS for z/OS, Version 3.1, this indirect support is removed. BTAM is no longer supported and all references to it have been removed.

This means that, if you have a network of BTAM terminals connected to a back-level CICS TOR, you will not be able (as you were in previous CICS releases) to route transactions from them to a CICS TS for z/OS, Version 3.1 application-owning region (AOR). You must either upgrade your terminals or route to a previous version of CICS.

Changes in CICS support for TCAM

CICS Transaction Server for z/OS, Version 3 Release 1 does not support the TCAM/ACB interface. It supports the TCAM/DCB interface indirectly.

For several CICS releases, the ACB interface of TCAM has been supported only indirectly: that is, by transaction routing from a back-level terminal-owning region (TOR) to which the terminals were attached. In CICS TS for z/OS, Version 3.1, this indirect support is removed.

In previous CICS releases, the DCB interface of TCAM has been fully supported. That is:

1. TCAM/DCB could be used to connect terminals to a current-level CICS TOR.
2. Transactions started by TCAM/DCB-connected terminals could be routed to a current-level CICS AOR.

In CICS TS for z/OS, Version 3.1, only the second operation is supported.

As a result of these changes:

* If you have a network of terminals connected by the ACB interface of TCAM to a back-level CICS TOR, you will not be able (as you were in previous CICS releases) to route transactions from them to a CICS TS for z/OS, Version 3.1 AOR. You must migrate your connections to use TCAM/DCB or (preferably) ACF/VTAM, or route to a previous version of CICS. (All terminals that support TCAM/ACB also support ACF/VTAM.)
* If you have a network of terminals connected by the DCB interface of TCAM to, for example, a CICS TS 2.3 TOR, you will not be able to migrate the TOR to CICS TS for z/OS, Version 3.1. To do so, you must migrate your connections to use ACF/VTAM.

- If you have a network of terminals connected by the DCB interface of TCAM to a back-level CICS TOR, you will (as in previous CICS releases) be able to route transactions from them to a CICS TS for z/OS, Version 3.1 AOR. However, you are recommended to migrate your connections to use ACF/VTAM.

Changes to CICS externals

Changes to system initialization parameters

The **TCAM={NO | YES}** system initialization parameter is now obsolete and is retained only for compatibility with previous CICS releases. If it is specified, it is rejected with a message and TCAM=NO is assumed.

Changes to resource definition

CICS no longer supports BTAM terminals, even indirectly. Thus you can no longer define BTAM terminals, even as remote resources.

CICS no longer supports local TCAM terminals. Therefore the following resource definition macros can no longer be used to define local TCAM terminals:
- DFHTCT TYPE=SCSDI
- DFHTCT TYPE=LINE
- DFHTCT TYPE=TERMINAL

It is still possible to define remote TCAM terminals. You can do this using either of the following:
- A single DFHTCT TYPE=REMOTE macro.
- A DFHTCT TYPE=REGION macro, followed by a DFHTCT TYPE=LINE and a DFHTCT TYPE=TERMINAL macro. CICS uses only the remote attributes of the DFHTCT TYPE=LINE and DFHTCT TYPE=TERMINAL macros.

Changes to the application programming interface

The following, BTAM-related, **EXEC CICS API** commands are obsolete:
- CONVERSE (SYSTEM/3)
- CONVERSE (SYSTEM/7)
- CONVERSE (2741)
- CONVERSE (2770)
- CONVERSE (2780)
- CONVERSE (3600 BTAM)
- CONVERSE (3735)
- CONVERSE (3740)
- ISSUE COPY (3270 display)
- RECEIVE (SYSTEM/3)
- RECEIVE (SYSTEM/7)
- RECEIVE (2741)
- RECEIVE (3600 BTAM)
- RECEIVE (3735)
- RECEIVE (3740)
- SEND (SYSTEM/3)
- SEND (SYSTEM/7)
- SEND (2741)
- SEND (3600 BTAM)
- SEND (3735)
- SEND (3740)

Changes to global user exits

The following global user exits in the terminal control program (which were invoked on TCAM input and output events) are no longer called:
- XTCTIN
- XTCTOUT

Changes to user-replaceable programs

Because local TCAM terminals are no longer supported, the terminal error program is not invoked for TCAM terminals. It is still invoked for sequential devices.

Changes to sample programs

The DFHSPTM1 and DFHSPTM2 sample TCAM programs are no longer supplied with CICS.

Migration

If you have a network of BTAM terminals connected to a back-level CICS terminal-owning region (TOR), you will not be able (as you were in previous CICS releases) to route transactions from them to a CICS TS for z/OS, Version 3.1 application-owning region (AOR). You must either upgrade your terminals or route to a previous version of CICS.

If you have a network of terminals connected by the ACB interface of TCAM to a back-level CICS TOR, you will not be able (as you were in previous CICS releases) to route transactions from them to a CICS TS for z/OS, Version 3.1 AOR. You must migrate your connections to use TCAM/DCB or (preferably) ACF/VTAM, or route to a previous version of CICS. (All terminals that support TCAM/ACB also support ACF/VTAM.)

If you have a network of terminals connected by the DCB interface of TCAM to, for example, a CICS TS 2.3 TOR, you will not be able to migrate the TOR to CICS TS for z/OS, Version 3.1. To do so, you must migrate your connections to use ACF/VTAM.

If you have a network of terminals connected by the DCB interface of TCAM to a back-level CICS TOR, you will (as in previous CICS releases) be able to route transactions from them to a CICS TS for z/OS, Version 3.1 AOR. However, you are recommended to migrate your connections to use ACF/VTAM.

Coexistence

CICS TS for z/OS, Version 3.1 does not support transaction routing or function shipping from BTAM terminals attached to a pre-CICS TS 3.1 terminal-owning region.

CICS TS for z/OS, Version 3.1 does not support transaction routing or function shipping from terminals attached by TCAM/ACB to a pre-CICS TS 3.1 terminal-owning region.

CICS TS for z/OS, Version 3.1 *does* support transaction routing and function shipping from terminals attached by TCAM/DCB to a pre-CICS TS 3.1 terminal-owning region.

Chapter 22. Withdrawal of support for 1-byte console id

Support for defining terminals using the 1-byte console id is withdrawn. The CONSOLE attribute on the TERMINAL resource definition is obsolete, but is supported to provide compatibility with earlier releases of CICS.

You can define terminals using the CONSNAME(*name*) attribute on the TERMINAL resource definition.

CICSPlex SM support

The 1-byte console ID is no longer in use, and support for it has been removed.

Changes to CICSPlex SM application programming interface

TERMDEF resource table

The CONSOLE attribute in the TERMDEF resource table is no longer valid in CICS Transaction Server for z/OS, Version 3 Release 1 or later releases.

Changes to CICSPlex SM end user interface views

The following change has been made:

TERMDEF view

The TERMDEF view remains unchanged although the following attribute is no longer valid in CICS Transaction Server for z/OS, Version 3 Release 1, or later releases.

CONSOLE
Consol ID

Changes to CICSPlex SM Web User Interface

The following change has been made:

Terminal Definition view

The TERMDEF (EYUSTARTTERMDEF) view remains unchanged although the following attribute is no longer valid in CICS Transaction Server for z/OS, Version 3 Release 1, or later releases.

CONSOLE
Consol ID

Messages

BBMZA094E CAS (SSID) INVALID REPLY

\# The removal of the 1-byte console ID means that CICSPlex SM message
\# BBMZA094E now goes to all active consoles, rather than simply the console that
\# replied to message BBMZA094E. The text of the message is unchanged.

Chapter 23. Withdrawal of the CICS Connector for CICS TS

Support for the CICS Connector for CICS TS, introduced in CICS TS for z/OS, Version 2.1, is withdrawn.

A CICS connector is a software component that allows a Java client application to invoke a CICS application. CICS TS for z/OS, Version 2.3 introduced a new CICS connector, the CCI Connector for CICS TS, that performs a similar role to the CICS Connector for CICS TS—that is, it enables a Java program or enterprise bean running on CICS Transaction Server for z/OS to link to a CICS server program. However, whereas the old CICS Connector for CICS TS implemented the IBM-proprietary Common Connector Framework (CCF) interface, the new CCI Connector for CICS TS implements the industry-standard Common Client Interface (CCI) defined by the J2EE Connector Architecture Specification, Version 1.0.

Since CICS TS for z/OS, Version 2.3 it has been recommended that:
- When writing new connector applications, you use the CCI Connector for CICS TS rather than the CICS Connector for CICS TS
- You migrate any existing applications that use the CICS Connector for CICS TS to use the CCI Connector for CICS TS instead

Because runtime support for the CICS Connector for CICS TS is withdrawn in CICS TS for z/OS, Version 3.1, these recommendations have now become mandatory.

Note: In previous releases, it was possible to program the CICS Connector for CICS TS in either of two ways: using the high-level CCF API or the lower-level, CICS-specific, external call interface (ECI) of the CICS Transaction Gateway API. The ECI base classes are no longer supplied with CICS.

For advice on using the CCI Connector for CICS TS in new applications, and on migrating existing applications that use the CICS Connector for CICS TS to use the CCI Connector for CICS TS instead, see *Java Applications in CICS.*

Chapter 24. Withdrawal of run-time support for Java program objects and hot-pooling

Run-time support for Java program objects and for hot-pooling (HPJ) is withdrawn.

In CICS TS 1.3, as an alternative to running Java programs in a Java Virtual Machine (JVM), VisualAge® for Java, Enterprise Edition for OS/390 (ET/390) could be used to bind Java bytecode into OS/390 executable files, known as Java program objects. The Java program objects were stored in OS/390 PDSE libraries and executed by CICS in a Language Environment run-unit, or enclave. The Java run-time component of ET/390 provided this run-time support in the CICS region.

The Language Environment enclave could be built and initialized for each invocation, in which case the Java program object was executed under the QR TCB. Alternatively, to reduce performance overheads, a preinitialized and persistent enclave could be reused for multiple invocations of the program. This feature was known as hot-pooling. When hot-pooling was specified for a Java program object, CICS used the PIPI preinitialization services of z/OS Language Environment to build the enclave, and executed the Java program object in the CICS region under the control of an open transaction environment (OTE) task control block (TCB) in H8 mode.

In CICS TS for z/OS, Version 2.3, run-time support was provided for existing hpj-compiled Java program objects, but no support was provided for developing new Java program objects nor for modifying existing Java program objects. In CICS TS for z/OS, Version 3.1, run-time support for hpj-compiled Java program objects is withdrawn.

You must migrate any hpj-compiled Java program objects to run in a Java Virtual Machine (JVM). The *CICS Migration Guide* explains how to do this. For information about the CICS JVM and about Java programming for CICS, see *Java Applications in CICS.*

If you attempt to execute a Java program object in CICS TS for z/OS, Version 3.1, an ALIG abend is issued.

The open TCB mode H8, which was used for hot-pooling Java program objects, no longer exists.

Changes to CICS externals

Changes to system initialization parameters

The system initialization parameter MAXHPTCBS is removed. MAXHPTCBS controlled the open TCB mode H8.

Changes to resource definition

- The HOTPOOL attribute is removed from the PROGRAM resource definition. The attribute was used to specify whether or not the Java program object was to be run in a preinitialized Language Environment enclave reused by multiple invocations of the program, under control of an H8 TCB.
- The sample application program group DFH$JAVA is removed. This group contained the resource definitions needed for the sample applications for Java

support using VisualAge for Java, Enterprise Edition for OS/390. The same sample applications are defined for use with a JVM by the DFH$JVM group.

- DFHTASK field 278, CICS MAXHPTCBS delay time, is removed from the DFHMCT TYPE=RECORD macro.

Changes to the application programming interface

The RESP2 value 43, which was used to qualify the INVREQ response to EXEC CICS LINK and XCTL commands, and to the BTS commands LINK ACQPROCESS and LINK ACTIVITY, is removed.

43   A LINK (or an XCTL) has been attempted to a hot-pooled Java program object while there is already a hot-pooled program on the link stack.

Changes to the system programming interface

- The HOTPOOL option is removed from the **EXEC CICS CREATE PROGRAM** command.
- The HOTPOOLING option is removed from the **EXEC CICS INQUIRE PROGRAM** command.
- The HOTPOOLING option is removed from the **EXEC CICS SET PROGRAM** command.
- The ACTHPTCBS and MAXHPTCBS options are removed from the **EXEC CICS INQUIRE DISPATCHER** command. These options were used to inquire on the number of H8 mode open TCBs currently allocated to user tasks, and the maximum number of H8 mode open TCBs that CICS was allowed to attach and maintain.
- The MAXHPTCBS option is removed from the **EXEC CICS SET DISPATCHER** command.
- The CVDAs HOTPOOL (1065) and NOTHOTPOOL (1066) are deleted.

Changes to CEMT

- The HOTPOOLING field is removed from the **CEMT INQUIRE PROGRAM** display.
- The HOTPOOL and NOTHOTPOOL options are removed from the **CEMT SET PROGRAM** command.
- The ACTHPTCBS and MAXHPTCBS fields are removed from the **CEMT INQUIRE DISPATCHER** display.
- The MAXHPTCBS option is removed from the **CEMT SET DISPATCHER** command.

Changes to global user exits

The global user exit task indicator field, addressed by UEPGIND, which is part of the DFHUEPAR standard parameter list, no longer includes the symbolic value UEPTH8. UEPTH8 represented the open TCB mode H8.

Changes to the exit programming interface (XPI)

The HOTPOOL option is removed from the DFHPGISX calls **INQUIRE PROGRAM** and **SET PROGRAM**.

Changes to user-replaceable programs

The user-replaceable programs DFHAPH8O and DFHJHPAT are removed.

- DFHAPH8O was provided to allow you to alter the default Language Environment run-time options for the Language Environment enclave where a Java program object was to be run.

- DFHJHPAT was optional and could be used for your own purposes, such as tracing. It was called before a Java program object was invoked.

Changes to monitoring

The monitoring data field 278 in group DFHTASK is removed. The open TCB mode H8, which was used for hot-pooling Java program objects and was controlled by MAXHPTCBS, no longer exists.

Changes to statistics

The TCB mode H8 is no longer displayed in the TCB Mode statistics. TCB Mode statistics are mapped by the DSGTCBM DSECT within the DFHDSGDS DSECT.

Changes to problem determination

- The trace entries for Java hot-pooling (AP 19A0 to AP 19C4) are removed.
- Messages DFHAP1219 to DFHAP1225 are removed.
- Abends AJH0 to AJHF are removed.
- The resource type HP_POOL is no longer a cause of dispatcher waits.

If you attempt to execute a Java program object in CICS TS for z/OS, Version 3.1, an ALIG abend is issued.

CICSPlex SM support

The removal of run-time support for Java program objects and for hot-pooling (HPJ) have resulted in a number of changes to the CICSPlex SM interfaces.

Changes to CICSPlex SM end user interface views

There are no new end user interface views in CICS Transaction Server for z/OS, Version 3 Release 1. However, changes have been made to the following existing screens:

- PROGDEF - The attribute HOTPOOLING is no longer valid but is still displayed.
- PROGRAM - The attribute HOTPOOLING has been removed.

Changes to the CICSPlex SM application programming interface

The removal of the run-time support for Java program objects and for hot-pooling (HPJ) has resulted in the following changes:

- "CICSRGN resource table"
- "PROGRAM resource table"
- "PROGDEF resource table"

CICSRGN resource table

The existing **HOTPOOLING** statistic is no longer valid in CICS TS 3.1.

PROGRAM resource table

The SPI attribute HOTPOOLING is no longer valid in CICS TS 3.1.

PROGDEF resource table

The attribute HOTPOOLING is no longer valid in CICS TS 3.1.

Changes to the CICSPlex SM Web User Interface

Changes have been made to the following views:

* "Program Definition view"
* "Program view"

Program Definition view

The attribute HOTPOOLING is no longer valid in CICS TS for z/OS, V3.1 but the PROGDEF (EYUSTARTPROGDEF.DETAILED) view remains the same.

Program view

The attribute HOTPOOLING is no longer valid in CICS TS for z/OS, V3.1 but the PROGRAM (EYUSTARTPROGRAM.DETAILED) view remains the same.

Chapter 25. Removal of CICSPlex SM support for Windows remote MAS

Previous releases of CICSPlex SM have supported the CICS for Windows component of TXSeries, Version 4.3.0.4 and TXSeries, Version 5.0 (also known as NT 4.3 and NT 5.0) in the management of a remote managed application system (RMAS). This support is no longer necessary and the CICSPlex SM TXSeries agent has been removed for CICS Transaction Server for z/OS, Version 3.1 and later releases. Therefore, it is no longer possible to set up a CICSPlex SM remote MAS agent for Windows.

Customers who wish to do so can continue to use the CICS Transaction Servers 2.3 or 2.2 for TXSeries support in CICSPlex SM.

Chapter 26. Withdrawal of the CICS Transaction Affinities Utility

The CICS Transaction Affinities Utility is not supplied with CICS TS for z/OS, Version 3.1. Its functions of detecting and reporting transaction affinities are now provided by the CICS Interdependency Analyzer, which is a more sophisticated tool.

Using the CICS Interdependency Analyzer, you can:

- Identify the sets of resources used by individual CICS transactions, and their relationships to other resources. This enables you to understand the make-up of your application set: you can see what a CICS region contains; what resources a transaction needs in order to run; which programs use which resources; and which resources are no longer used. Thus your ability to maintain, enhance, modify, or redistribute your applications is much improved.
- Identify possible transaction affinities. Affinities require particular groups of transactions to be run either in the same CICS region, or in a particular region. The ability to identify transaction affinities is useful in a dynamic routing environment: you need to know of any restrictions that prevent particular transactions being routed to particular application-owning regions (AORs); or that require particular transactions to be routed to particular AORs.

For more information about the CICS Interdependency Analyzer, see the *CICS Interdependency Analyzer User's Guide and Reference*.

Part 7. General Information

Chapter 27. The CICS operating environment

This topic gives some information about related products that you need in order to use the CICS and CICSPlex SM elements of CICS Transaction Server for z/OS.

Hardware requirements

Processors

The basic requirement is for a processor that supports the prerequisite operating system and has sufficient processor storage to meet the requirements of z/OS V1.4, CICS TS for z/OS, Version 3.1, the application programs, the access methods, and all other software being run. This includes the IBM eServer™ zSeries 990.

Parallel Sysplex® support

A Parallel Sysplex environment is required by each of the data-sharing facilities supported by CICS, and by the MVS system logger's log stream merging facility. This requires:

- One or more coupling facilities with their associated coupling links installed
- An IBM sysplex timer to provide a common external time source
- Sufficient DASD paths to support the number of central processor complexes (CPCs) in the sysplex. The DASD paths can be provided either by DASD controllers with enough paths to dedicate one to each CPC in the sysplex, or by an ESCON® director.

CICS support for data sharing can be used to access data in IMS databases, DB2 databases, VSAM data sets, CICS temporary storage, coupling facility data tables, and named counters.

Cryptographic hardware

zSeries cryptographic hardware is required:

- To exploit the WS-Security capability.
- To fully benefit from the performance improvements to SSL encryption.

Both functions rely upon the z/OS Integrated Cryptographic Services Facility (ICSF).

Katakana Terminal Devices

Because CICS has to issue certain messages in mixed-case, the product is not supported with displays or terminal emulators that are restricted to the non-extended single-byte character set (SBCS) Katakana part of code page 930.

Software Requirements

Note that the *Program Directory* (GI10-6427) will normally contain the most up-to-date information on software requirements.

Operating environment

CICS TS for z/OS, Version 3.1 requires z/OS V1.4, or later. Note that it will not initialize in an environment with a lower level of operating system installed.

The Language Environment library SCEERUN must be available to CICS during CICS initialization, by inclusion in either the STEPLIB concatenation or the LNKLIST. Language Environment services are used by a number of CICS functions.

For Java application programs or enterprise beans, the IBM SDK for z/OS, Java 2 Technology Edition, featuring persistent reusable JVM technology, Version 1.4, program 5655-I56. This must be at the V1.4.2 level.

- The IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4, is available, at no charge, on tape or by download from: http://www.s390.ibm.com/java/.
- Note that IBM 64-Bit SDK For z/OS, Java 2 Technology Edition, Version 1.4, program number 5655-M30, does not provide the required function.

For WS-Security support, the IBM XML Toolkit for z/OS V1.9 is required. This is a no-charge product, program number 5655-J51. You can download the toolkit from the following site: http://www.ibm.com/servers/eserver/zseries/software/xml/.

- You must install version 1.9. Later versions do not work with Web Services Security support in CICS.
- The toolkit is an MVS feature and should be installed in the SMP/E zone for MVS.
- Specify a valid keyring on the `KEYRING` system initialization parameter.
- Apply the PTFs for APARs PK65352 and PK97657 to CICS, which change the required version of the toolkit from V1.7 to V1.9.

For deployment of enterprise beans, WebSphere Application Server V5.0, or later, is required.

- The component to be used is the Application Server Toolkit (ASTK) for Windows. Note that the Application Assembly Tool (AAT), provided with early deliveries of V5.0, is not supported.
- Note also that the ASTK is also included in WebSphere Studio Enterprise Developer V5.1.

JNDI support for enterprise beans can be provided by the LDAP server provided in SecureWay Security Server and licensed as part of the base z/OS operating system. CICS TS V3.1 will interoperate with WebSphere Application Server (any platform) V5 and V6. This applies directly for customers using RMI/IIOP for interoperability; and via CICS Transaction Gateway V5.0 or later for those using JCA.

For developing Java programs (including enterprise beans) for use with CICS TS V3.1, the members of the WebSphere Studio family V5, and Rational® Application Developer V6, are supported.

The System SSL Security Level 3 feature for z/OS is required to use cipher suites with 128-bit encryption or above.

Other supported products

The following levels of other products are supported for use with CICS TS for z/OS, Version 3.1:

- IMS Database Manager V7 (5655-B01)
- IMS Database Manager V8 (5655-C56)
- IMS Database Manager V9 (5655-J38)
- DB2 Universal Database™ Server for OS/390 V6.1 (5645-DB2)

- For SQLJ/JDBC support, with PTF for APAR PQ84783
  - Does not support DB2 Group Attach
- DB2 Universal Database Server for OS/390 V7.1 (5675-DB2)
  - For SQLJ/JDBC support, with PTFs for APARs PQ84783 and 86525
  - For DB2 Group Attach, with APARs PQ44614, PQ45691, and PQ45692
- DB2 Universal Database for z/OS V8.1 (5625-DB2)
  - For SQLJ/JDBC support, with PTFs for APARs PQ84783 and 86525
- WebSphere MQ for z/OS V5.3 (5655-F10)
- Tivoli® Decision Support for OS/390 (5698-ID9) V1.7, with PTF for APAR
  PK07001 applied
- Tivoli Business Systems Manager V3.1
- CICS Universal Client Version 5.0, or later
- CICS Transaction Gateway Version 5.0, or later

Information Center environment

The Information Center, as a server, is supported on:
- Windows 2000 and Windows XP
- AIX® V5.1, or later
- Linux on Intel (RedHat and Suse)

For browsing the Information Center, you will need a browser that supports HTML
4.0 and the Document Object Model (DOM) standard. Suitable browsers include:

   Microsoft Internet Explorer Version 6.0

   Netscape Navigator V7.0

   Mozilla V1.0

running on Windows 2000 or Windows XP.

To read PDF files shipped with the Information Center, you will need Adobe Acrobat
Reader 5.0 or 6.0. The files have been generated using Adobe Acrobat Distiller 6.0
at the Acrobat 6.0 (PDF 1.5) level. They can be read using Adobe Acrobat Reader
5.0, but Reader 6.0 is necessary if you need the accessibility features of Distiller
6.0.

Support for CICS Tools and related products

CICS Interdependency Analyzer

The following can be used with CICS TS for z/OS, Version 3.1:
   CICS Interdependency Analyzer for z/OS V1.3, with PTF for APAR PQ95065.

The following do **not** run with CICS TS for z/OS, Version 3.1:
   CICS Interdependency Analyzer for z/OS V1.1.
   CICS Interdependency Analyzer for z/OS V1.2.

CICS Performance Analyzer

CICS Performance Analyzer for z/OS V1.3 does **not** support SMF 110 data from
CICS TS for z/OS, Version 3.1.

CICS Performance Monitor

CICS Performance Monitor for z/OS V1.1 does **not** support CICS TS for z/OS, Version 3.1.

CICS Performance Monitor for z/OS V1.2, with service applied, provides toleration support for CICS TS for z/OS, Version 3.1, at the CICS TS for z/OS, Version 2.3 level.

Tivoli OMEGAMON®

The following support CICS TS for z/OS, Version 3.1. However, this support does not include exploitation of the new function of CICS TS for z/OS, Version 3.1.
   Tivoli OMEGAMON II for CICS V520.
   Tivoli OMEGAMON XE for CICS V100.
   Tivoli OMEGAMON XE for CICSplex V220, with service applied.

CICS VSAM Recovery

The following provides recovery support for VSAM files processed by CICS TS for z/OS, Version 3.1:
   CICS VSAM Recovery V3.2.
   CICS VSAM Recovery V3.3.

CICS Business Event Publisher

CICS Business Event Publisher for MQSeries® Version 1.2, with service applied, can be used with CICS TS for z/OS, Version 3.1.

CICS Online Transmission Time Optimizer

CICS Online Transmission Time Optimizer for z/OS V1.1 can be used with CICS TS for z/OS, Version 3.1.

Session Manager

The following can be used with CICS TS for z/OS, Version 3.1:
   Session Manager for z/OS V1.1.
   Session Manager for z/OS V1.2.

CICS VSAM Transparency

CICS VSAM Transparency for z/OS V1.1 can be used with CICS TS for z/OS, Version 3.1.

CICS VSAM Copy

CICS VSAM Copy for z/OS V1.1, with service applied, can be used with CICS TS for z/OS, Version 3.1.

CICS Batch Application Control

CICS Batch Application Control for z/OS V1.1, with service applied, can be used with CICS TS for z/OS, Version 3.1.

MQSeries Integrator Agent

MQSeries Integrator Agent for CICS Transaction Server V1.1 does **not** run with CICS TS for z/OS, Version 3.1.

Fault Analyzer

The following can be used with CICS TS for z/OS, Version 3.1:
 Fault Analyzer for z/OS and OS/390 V3.1, with PTF UQ77156 for APAR PQ74048.
 Fault Analyzer for z/OS V4.1.
 Fault Analyzer for z/OS V5.1.

Debug Tool

The following can be used with CICS TS for z/OS, Version 3.1:
\# Debug Tool for z/OS V5.1, with PTF UQ88297 for APAR PQ94401.

Compatibility

z/OS conversion services

Unlike previous levels of CICS Transaction Server, CICS TS V3 can use z/OS services to perform conversions beyond those supported by CICS TS in previous releases. An example is conversions to and from Unicode, which might be required to support Web services. This requires z/OS to have the initial conversion image installed, which can only be done on a system IPL. If it is wished to install CICS TS V3 without a re-IPL of z/OS, this can be done provided the initial conversion image is installed during a previous system IPL. The conversion image does not include any code from CICS TS; it can also be refreshed without any need for a further IPL.

JVM modes in CICS

Customers using Java programs in CICS TS V3.1 are recommended to use continuous mode. Support for continuous mode was introduced in CICS TS V2.3; in order to bring CICS use of Java into line with standard practices, support for resettable mode will be removed in a future release of CICS TS.

\# ### SOAP for CICS feature

\# Applications developed to use the SOAP for CICS feature will continue to run with
\# CICS TS V3.1. However, customers are recommended to migrate to the Web
\# services support capabilities of CICS TS V3.1 because support for the SOAP for
\# CICS Feature support will cease when support of CICS TS V2.3 ceases.

Common Connector Framework (CCF)

The Common Connector Framework (CCF), which was the predecessor interface to the Common Client Interface (CCI), is not supported by CICS TS V3.1. The intention to remove this support was indicated in the announcement of CICS TS V2.3.

ECI Base Classes (ECIREQUEST)

The ECI Base Classes (ECIREQUEST, which were introduced for compatibility with the CICS Transaction Gateway), are not included in CICS TS V3.1. The

recommended replacement is the COMMON CLIENT INTERFACE CONNECTOR FOR CICS TS (CCI Connector for CICS TS), introduced in CICS TS V2.3, when it was announced that ECIREQUEST would be removed.

Transaction Affinities utility

CICS TS V3.1 does not include the detector and reporter components previously provided as part of the CICS Transaction Affinities utility. These components are now incorporated in IBM CICS Interdependency Analyzer for z/OS V1.3, announced in August 2004, which has the capability of analyzing both interdependencies and affinities. The load library scanner component of the CICS Transaction Affinities utility remains in CICS TS V3.1, and can produce reports on application programs which have potential affinities.

Chapter 28. Threadsafe application programming interface commands

Most new commands in CICS Transaction Server for z/OS, Version 3 Release 1 are threadsafe. Additionally, some existing commands have been made threadsafe in this release.

New commands that are threadsafe
> CONVERTTIME
> DELETE CONTAINER (CHANNEL)
> GET CONTAINER (CHANNEL)
> INVOKE WEBSERVICE
> MOVE CONTAINER
> PUT CONTAINER (CHANNEL)
> SOAPFAULT ADD
> SOAPFAULT CREATE
> SOAPFAULT DELETE
> WEB CONVERSE
> WEB CLOSE
> WEB OPEN
> WEB PARSE URL
> WEB RECEIVE (Client)
> WEB SEND (Client)

Existing commands that are now threadsafe
> WEB ENDBROWSE FORMFIELD
> WEB ENDBROWSE HTTPHEADER
> WEB EXTRACT
> WEB READ FORMFIELD
> WEB READ HTTPHEADER
> WEB READNEXT FORMFIELD
> WEB READNEXT HTTPHEADER
> WEB RECEIVE (Server)
> WEB RETRIEVE
> WEB SEND (Server)
> WEB STARTBROWSE FORMFIELD
> WEB STARTBROWSE HTTPHEADER
> WEB WRITE HTTPHEADER

New commands that are not threadsafe
> START CHANNEL

Chapter 29. High-level language support

This reference topic describes the high-level programming languages supported by CICS, and provides information about which release of each language is supported in current releases of CICS.

COBOL Compilers

Compiler	Program number	Compiler in service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1 on page 364)	Use with WebSphere Studio Enterprise Developer
OS/VS COBOL	5740-CB1	No	**CICS TS V1.3:** Supported **CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** Not supported	**CICS TS V1.3, CICS TS V2.2, CICS TS V2.3:** The Language Environment component of z/OS is required; applications will run unchanged. **CICS TS V3.1:** Not supported	No	No
VS COBOL II	5668-023 5668-958	No	**CICS TS V1.3:** Supported with the `COBOL2` option **CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** Supported with the `COBOL2` and `COBOL3` options.	**CICS TS V1.3, CICS TS V2.2:** The Language Environment component of z/OS is required; applications will run unchanged. **CICS TS V2.3, CICS TS V3.1:** The Language Environment component of z/OS is required; CICS will use the Language Environment runtime exclusively. Application behavior might change.	Yes, with restrictions	No
SAA AD/Cycle® COBOL/370	5688-197 5668-958	No	**CICS TS V1.3:** Supported with the `COBOL2` option **CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** Supported with the `COBOL2` and `COBOL3` options.	Language Environment	Yes, with restrictions	No
COBOL for MVS and VM	5688-197	No	**CICS TS V1.3:** Supported with the `COBOL2` option **CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** Supported with the `COBOL2` and `COBOL3` options.	Language Environment	Yes, with restrictions	No

Compiler	Program number	Compiler in service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1)	Use with WebSphere Studio Enterprise Developer
COBOL for OS/390 and VM V2	5648-A25	Yes	**CICS TS V1.3:** Supported with the `COBOL2` option **CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** Supported with the `COBOL2` and `COBOL3` options.	Language Environment	Yes, with restrictions	No
COBOL for OS/390 and VM V2	5648-A25 (with PTF for APAR PQ45462)	Yes	Can use the integrated translator (see note 2)	Language Environment	Yes, with restrictions	Yes, with restrictions
Enterprise COBOL for z/OS and OS/390 V3	5655-G53	Yes	Can use the integrated translator (see note 2)	Language Environment	Yes	Yes

Notes:

1. IBM Distributed Debugger is available as a component of WebSphere Studio Enterprise Developer V5, and other IBM products.

   For more information, refer to: http://www.ibm.com/software/awdtools/debugger/.

\# 2. The integrated translator function requires IBM COBOL for OS/390 and
\# VM Version 2 Release 2, with PTF for APAR PQ45462, or Enterprise
\# COBOL for z/OS and OS/390 Version 3.

PL/I compilers

Compiler	Program number	Compiler in service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1 on page 365)	Use with WebSphere Studio Enterprise Developer
OS PL/I Optimizing Compiler V1	5724-PLI	No	Yes	**CICS TS V1.3, CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** The Language Environment component of z/OS is required; applications will run unchanged.	Yes, with restrictions	No
OS PL/I Optimizing Compiler V2	5668-909 5668-910 5668-911	No	Yes	**CICS TS V1.3, CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** The Language Environment component of z/OS is required; applications will run unchanged.	Yes, with restrictions	No
SAA AD/Cycle PL/I for MVS and VM	5688-235	No	Yes (see note 2 on page 365)	Language Environment	Yes, with restrictions	No

Compiler	Program number	Compiler in service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1)	Use with WebSphere Studio Enterprise Developer
PL/I for MVS and VM V1	5688-235	No	Yes (see note 2)	Language Environment	Yes, with restrictions	No
VisualAge PL/I for OS/390 V2	5655-B22	No	Yes (see note 2)	Language Environment	Yes, with restrictions	No
Enterprise PL/I for z/OS and OS/390 V3	5655-H31	Yes	Can use integrated translator (see note 2)	Language Environment	Yes	Yes

Notes:

1. IBM Distributed Debugger is available as a component of WebSphere Studio Enterprise Developer V5, and other IBM products.

   For more information, refer to: http://www.ibm.com/software/awdtools/debugger/.

2. The integrated translator function requires VisualAge PL/I for 3 OS/390, Version 2 Release 2.1, with PTF for APAR PQ45562, or Enterprise PL/I for z/OS and OS/390 Version 3.

C and C++ Compilers

Compiler	Program number	Compiler in service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1 on page 366)	Use with WebSphere Studio Enterprise Developer
C/370™ V1	5688-040	No	Yes	**CICS TS V1.3, CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** The Language Environment component of z/OS is required; applications will run unchanged.	Yes, with restrictions	No
C/370 V2	5688-187 5688-188	No	Yes	**CICS TS V1.3, CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** The Language Environment component of z/OS is required; applications will run unchanged.	Yes, with restrictions	No
SAA AD/Cycle C/370	5688-216	No	Yes	Language Environment	Yes, with restrictions	No
C/C++ for MVS/ESA	5655-121	No	Yes	Language Environment	Yes, with restrictions	No
C/C++ for OS/390	Component of 5647-A01	Yes	Yes	Language Environment	Yes, with restrictions	No
C/C++ for z/OS and OS/390	Component of 5694-A01	Yes	Yes	Language Environment	Yes	No

Compiler	Program number	Compiler in service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1)	Use with WebSphere Studio Enterprise Developer
z/OS V1.7 XL C/C++	5694-A01	Yes	Yes. The compiler provides support for the CICS integrated translator	Language Environment	Yes	No

Notes:

1. IBM Distributed Debugger is available as a component of WebSphere Studio Enterprise Developer V5, and other IBM products.

   For more information, refer to: http://www.ibm.com/software/awdtools/debugger/.

Java Support

Compiler or JVM	Program number	In service	CICS translator support	CICS run time support	Use of IBM Distributed Debugger (see note 1 on page 367)	Use with WebSphere Studio Enterprise Developer
VisualAge for Java, Enterprise Edition V2 - Enterprise Toolkit for OS/390 (see note 2 on page 367)	5655-JAV	No	**CICS TS V1.3, CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** No translator required - use the JCICS classes.	Language Environment	Yes, with restrictions	No
Java for OS/390 at SDK 1.1.8		Yes	**CICS TS V1.3 only:** No translator required - use the JCICS classes. **CICS TS V2.2, CICS TS V2.3, CICS TS V3.1:** Not supported	Language Environment	Yes	Yes
Developer Kit for OS/390, Java 2 Technology Edition, V1.3.1	5655-D35	Yes	**CICS TS V2.2 only:** No translator required - use the JCICS classes. **CICS TS V1.3, CICS TS V2.3, CICS TS V3.1:** Not supported	Language Environment	Yes	Yes
SDK for z/OS, Java 2 Technology Edition, V1.4.2	5655-I56	Yes	**CICS TS V2.3, CICS TS V3.1:** No translator required - use the JCICS classes. **CICS TS V1.3, CICS TS V2.2:** Not supported	Language Environment	Yes	Yes

Notes:

1. IBM Distributed Debugger is available as a component of WebSphere Studio Enterprise Developer V5, and other IBM products.

   For more information, refer to: http://www.ibm.com/software/awdtools/debugger/.

2. Java program objects are programs compiled with the VisualAge for Java Enterprise Toolkit for OS/390 (ET/390) byte-code binder (they are compiled with the hpj command, and are sometimes referred to as compiled Java programs or as HPJ programs).

Part 8. Publications

Chapter 30. The Eclipse information center

In CICS Transaction Server for z/OS, Version 3 Release 1, the CICS Information Center runs within the Eclipse help system, a framework that contains a number of documentation plug-ins that make up the information center. The new look and feel of the information center, in particular the welcome page, is now consistent with other product information centers.

The CICS TS 3.1. documentation is contained in a single plug-in. You also have the option of installing any or all of the CICS tooling products - there is one plug-in per tooling product. The information center, also known as a help system, runs in two modes. You can start it locally on your workstation, or you can start it as a server with remote access via a browser.

You can access information about the latest functionality in CICS, find out about highlights of the product and link to CICS resources on the Web from the new Welcome page.

Using the Contents pane, you can access information on CICS by functional area or by looking for a user goal that matches the task you are performing. You can also view the books in HTML and PDF by selecting *Library and PDFs*.

The following features are implemented in this release of the information center:
* Extension of platform support to Linux, Linux on zSeries and z/OS.
* What's New section
* Information roadmaps
* Learning paths
* Searching and lookups
* Troubleshooting and support
* Bookmarks
* User preferences
* Revised navigation

Benefits of the Eclipse information center

The move to an Eclipse framework brings the benefit of support for additional platforms. You can now run the information center locally on a Linux RedHat or Linux SuSE workstation. In addition, the information center now runs as a server on Linux RedHat for zSeries, Linux SuSE for zSeries and z/OS. This allows you greater flexibility for where you host your information center. For a complete list of supported platforms, see "Requirements" on page 372.

The installation of the information center has been greatly simplified, so there is no longer a GUI installation. The readme file that is provided with the information center explains how to install the Eclipse help system. The structure of Eclipse means that you can put multiple documentation plug-ins into one help system, including plug-ins from other products. This enables you to customize the information that is contained in your information center. In addition, you can integrate the CICS documentation plug-in into an Eclipse-based IDE.

Searching the CICS documentation has improved, as Eclipse uses the Lucene search engine. NetQuestion is no longer required. When you search in the information center, each instance of the string query is now highlighted in the HTML content.

A number of navigation improvements have been made to the information center. Two new features are learning paths and information roadmaps. A learning path is a sequence of topics that help you learn about new functions more quickly. An information roadmap is a single topic that contains a comprehensive set of links about a functional area of CICS. The information roadmaps are designed to help you find useful resources both in the information center and on the Web, as well as providing guidance on what information is provided at each link.

There is also a *Troubleshooting and Support* section in the navigation that provides you with help on how to search knowledge bases, get fixes from IBM and report problems. Of particular benefit, is the Web search topic that allows you to query the CICS support Web site to look for solutions to problems. This topic also includes a Google search facility. In addition, key technotes from multiple releases are provided to help you quickly troubleshoot problems.

Terminology

This topic contains the terminology used by the Eclipse information center.

plug-in
A self-contained software component that modifies (adds or changes) function in a particular software system. When a user adds a plug-in to a software system, the original software system remains intact. In the case of documentation plug-ins, the plug-in contains all of the files relating to product documentation and how it is viewed in Eclipse.

learning path
A sequence of information center topics that helps new users learn about a functional area of the product

information roadmap
A topic that contains a comprehensive set of links to information resources in the information center and on the Web, along with guidance on where each link takes the user.

local mode
The implementation of the information center on a workstation, using the *IC_local_start* file. The information center automatically launches in the system default browser.

server mode
The implementation of the information center on a server, using the *IC_server_start* file. The information center runs on a specific port, and users access the information center remotely using their browsers.

Requirements

The information center requires the following operating systems to run:
*   Windows 2000
*   Windows XP
*   AIX 5.2 and 5.3
*   Linux RedHat Enterprise 3.0

- Linux SuSE Enterprise 3.0
- Linux RedHat Enterprise 8 and 9 for zSeries
- Linux SuSE Enterprise 8 and 9 for zSeries
- z/OS 1.4 or later

Please note that support for the information center on Linux for zSeries and z/OS is only offered in server mode.

The CICS TS 3.1 Information Center uses a Java Runtime Environment (JRE). A JRE for each platform is provided with the information center, except for z/OS. If you want to run an information center on z/OS, you need to use the JRE provided with the operating system.

To get the best results when viewing the information center, it is recommended that you use one of the following browsers:
- Microsoft Internet Explorer 6.0
- Mozilla 1.7

What's New section

The *What's New* section is a new navigation structure that contains all of the information relating to the new release, and is fully integrated with the rest of the information center.

To access information that describes the new features of CICS, you can use the links on the welcome page of the information center as a starting point. The information is divided into the three themes of the release: CICS integration, Application transformation, and Enterprise management. You can also use the navigation by expanding the What's New section to find the particular function you are looking for.

Information Roadmaps

An information roadmap is a single topic that contains links to a variety of resources on a particular functional area of CICS.

Each roadmap is divided into sections, which are summarized in a table of contents at the top of the topic. Sections cover areas such as basic skills, installation, migration, administration, application development, and security. The links in each section indicate whether a resource is internal or external to the information center, as well as providing a brief summary of what the resource contains. Resources include information center topics, redbooks, tutorials, white papers and articles. At the end of each roadmap is an education section that contains further reading, links to courses and educational material.

There are three information roadmaps in the information center. They cover the following functions:
- Web Services
- CICSPlex System Manager
- Java support and programming in CICS

You can access the information roadmaps directly from the welcome page of the information center, or expand the navigation in the Contents pane.

Learning paths

A learning path is a set of topics that help you learn about an area of CICS.

Each learning path has an introduction that explains the objectives and the intended audience. A learning path is designed to be read sequentially, but the introduction contains all of the steps in the learning path, so that you can start at any point if you need to. Button navigation is provided at the bottom of each topic to move through the path. There are also links to further information, so that you can leave the path at any time to find out more about a particular function or concept. At the end of the learning path is a summary topic. It summarizes what you have learned and provides useful links to further reading and subsequent tasks in the information center.

The learning paths for this release cover the following areas:
* Web Services
* Installing CICSPlex SM: the WUI scenario

Techniques for searching in the information center

You can use several methods to search the information center, depending on the information that you are looking for and the mode that the information center is using to run.

To perform a search in the information center, enter a query in the **Search** field. The **Contents** pane displays the top 500 ranked results. Click on a search result to view the topic. The words from your query are highlighted.

To toggle between the navigation tree and the search results list, click the Contents tab ( ) or the Search results tab ( ) at the bottom on the **Contents** pane.

Searching for exact words or phrases

You can identify a search phrase as an exact string by enclosing it in double quotation marks. For example, "log file" searches for the string *log file*, not the separate words *log* and *file*. Without the quotation marks, the search term is interpreted as log AND file.

In English and German only, the search engine "stems" other forms of a single search word. For example, a search for the word *challenge*, will also find the word *challenging*. Use the double quotation marks when you do not want stemming.

Searching with wildcards

You can use the following wildcard characters:
* * Asterisk - for multiple unknown or variable characters in the term. For example the search term par* returns *partly*, *participate*, *partial*, and other words beginning with *par*.
* ? Question mark - for a single unknown or variable character. For example, the search term par? returns *part* but not *partial* or *partly*.

Searching with Boolean operators (AND, OR, NOT)

You can insert the binary operators: AND, OR, and NOT in your search term. For example:

database AND "log file" : To narrow your search to include topics that must contain both of the terms *database* and *"log file"*.

database OR "log file" :To widen your search for topics that contain either *database* or *"log file"*.

database NOT "log file" : Searches for topics that contain *database* without *log file*.

Narrowing your search scope

By default, all topics shown in the Contents pane are searched. However, you can narrow your searches to a particular set of topics. If you are running the information center locally, you can narrow your searches using a search list and save search lists to use again later.

To create a search list:
1. Click the **Search scope:** link next to the Search field. The Select Search Scope window opens.
2. Select **Search only the following topics** and select the New push button. The New Search List windows opens.
3. In the **Topics to search** list, select the navigation categories that you want to include in your search. You can expand categories to select only certain subcategories.
4. In the **List name:** text field, enter an appropriate name for your search list and select the **OK** button.

You can use the search list that you have created to search the topics for your search expression.

If you are running the information center on a server, you can narrow your searches to a particular section of the navigation using the **Advanced search:** link next to the Search field.

Searching for messages, commands, and parameters

To help you quickly find a specific reference item in the documentation, such as a message or command, a lookup facility is provided in the navigation. Select *Find commands, messages, parameters and more* in the navigation to use the lookup facility. This facility does not use the Lucene search engine, but instead uses an index. You can search in the following areas:
- API commands
- API parameters
- SPI commands
- SPI parameter
- CICS transactions
- Messages and codes
- Trace entries
- Resource definitions

- System initialization parameters
- Glossary terms
- Figure titles
- Table titles

Navigating the information center

To navigate around the information center, you can select or expand the links in the Contents pane. You can also use features such as synchronization and keyboard shortcuts.

The **Contents** pane ( ) on the left side of the help window displays topic titles in a table of contents or *navigation tree* structure. You can either click on the topic titles to display content in the main panel on the right, or you can expand the navigation to view the nested sections. If you opt to use the links in the HTML topics, you can use the **Back** ( ) and **Forward** ( ) buttons to navigate within the history of viewed topics.

Each HTML topic contains a set of links, located at the bottom of the topic, that are designed to help you navigate to related information. In addition, many of the topics contain button navigation that take you to the contents table of the book that the topic belongs to, the next and previous topic in the book and finally the index.

If you perform a search and want to return to the table of contents, select the Contents tab ( ) at the bottom of the pane. At any time, you can use the tabs at the bottom of the pane to move between the navigation and the results of the last search.

Synchronizing the table of contents

When you follow a link within a topic, the navigation tree does not automatically change to display and highlight the new topic. To see where the new topic fits in the navigation tree and synchronize the two views, click the **Refresh** button ( ) or the **Show in Table of Contents** ( ) button. The topic title for the currently displayed topic will be highlighted in the navigation tree.

Navigating using the keyboard

Use the following key combinations to navigate through the information center:
- To go to the next link, button or topic node from inside a panel in the information center, press Tab.
- To expand and collapse a node in the tree, press the Right and Left arrows.
- To move to the next topic node, press the Down arrow or Tab.
- To move to the previous topic node, press the Up arrow or Shift+Tab.
- To scroll all the way up or down, press Home or End.
- To go back, press Alt+Left arrow; to go forward, press Alt+Right arrow.
- To go to the next pane, press Ctrl+Tab.
- To move to the previous pane, press Shift+Ctrl+Tab.
- To print the active pane, press Ctrl+P.

Bookmarking a topic

There are two ways to bookmark topics in the information center. The method for bookmarking topics depends on whether you are running the information center locally or on a server.

When you are running the information center locally on your workstation, you can bookmark a selected topic by following these steps:

1. Select the Bookmark button above the main panel ( ) . This will place a bookmark in the Bookmarks panel.

2. Click on the Bookmarks view icon ( ) at the bottom of the Contents pane. The Contents pane will be replaced by the Bookmarks pane to display all of your bookmarks. The topic you selected will be at the bottom of the list of bookmarks.

If you want to delete the bookmark, select it and then use the Delete key.

When you are accessing the information center on a server, you can use the browser to bookmark a topic.

User Preferences

The user preferences allow you to customize certain aspects of the information center functionality.

The user preferences are interactive tables, accessible syntax diagrams and a text/image option. These preferences are now available from the *Product and information center overview* section of the navigation.

Chapter 31. The CICS Transaction Server for z/OS library

The CICS Transaction Server for z/OS Information Center is the primary source of user information for CICS Transaction Server. A small subset (the *entitlement set*) of the CICS TS publications is available as hardcopy.

The Information Center contains:
- Information for CICS Transaction Server in HTML format.
- Licensed and unlicensed CICS Transaction Server books provided as Adobe Portable Document Format (PDF) files. You can use these files to print hardcopy of the books.
- Information for related products in HTML format and PDF files.

One copy of the CICS Information Center, on a CD-ROM, is provided automatically with the product. Further copies can be ordered, at no additional charge, by specifying the Information Center feature number, 7014. You will also receive a small set of essential hardcopy books.

Licensed documentation is available only to licensees of the product. A version of the Information Center that contains only unlicensed information is available through the publications ordering system, order number SK3T-6945.

Books available as hardcopy

When you order CICS Transaction Server for z/OS, Version 3 Release 1, you will receive a small number of hardcopy books.

The hardcopy books are:
> *Memo to Licensees, GI10-2559*
> *CICS Transaction Server for z/OS Program Directory, GI10-2586*
> *CICS Transaction Server for z/OS Release Guide, GC34-6421*
> *CICS Transaction Server for z/OS Licensed Program Specification, GC34-6608*

You can order further copies of the following books, using the order number quoted above:
> *CICS Transaction Server for z/OS Release Guide*
> *CICS Transaction Server for z/OS Installation Guide*
> *CICS Transaction Server for z/OS Licensed Program Specification*

PDF-only books

The licensed and unlicensed CICS Transaction Server books are provided in the CICS Information Center as Adobe Portable Document Format (PDF) files. You can use these files to print hardcopy of the books.

CICS books for CICS Transaction Server for z/OS

General
> *CICS Transaction Server for z/OS Program Directory, GI10-2586*
> *CICS Transaction Server for z/OS Release Guide, GC34-6421*
> *CICS Transaction Server for z/OS Migration from CICS TS Version 1.3, GC34-6423*
> *CICS Transaction Server for z/OS Migration from CICS TS Version 2.2, GC34-6424*

CICS Transaction Server for z/OS Migration from CICS TS Version 2.3,
GC34-6425

CICS Transaction Server for z/OS Installation Guide, GC34-6426

Access to CICS

CICS Internet Guide, SC34-6450

CICS Web Services Guide, SC34-6458

Administration

CICS System Definition Guide, SC34-6428
CICS Customization Guide, SC34-6429
CICS Resource Definition Guide, SC34-6430
CICS Operations and Utilities Guide, SC34-6431
CICS RACF Security Guide, SC34-6454
CICS Supplied Transactions, SC34-6432

Programming

CICS Application Programming Guide, SC34-6433
CICS Application Programming Reference, SC34-6434
CICS System Programming Reference, SC34-6435
CICS Front End Programming Interface User's Guide, SC34-6436
CICS C++ OO Class Libraries, SC34-6437
CICS Distributed Transaction Programming Guide, SC34-6438
CICS Business Transaction Services, SC34-6439
Java Applications in CICS, SC34-6440
JCICS Class Reference, SC34-6001

Diagnosis

CICS Problem Determination Guide, SC34-6441
CICS Performance Guide, SC34-6452
CICS Messages and Codes, SC34-6442
CICS Diagnosis Reference, LY33-6110
CICS Recovery and Restart Guide, SC34-6451
CICS Data Areas, LY33-6107
CICS Trace Entries, SC34-6443
CICS Supplementary Data Areas, LY33-6108
CICS Debugging Tools Interfaces Reference, LY33-6109

Communication

CICS Intercommunication Guide, SC34-6448
CICS External Interfaces Guide, SC34-6449

Databases

CICS DB2 Guide, SC34-6457

CICS IMS Database Control Guide, SC34-6453

CICS Shared Data Tables Guide, SC34-6455

CICSPlex SM books for CICS Transaction Server for z/OS

General

CICSPlex SM Concepts and Planning, SC34-6459
CICSPlex SM User Interface Guide, SC34-6460
CICSPlex SM Web User Interface Guide, SC34-6461

Administration and Management
CICSPlex SM Administration, SC34-6462
CICSPlex SM Operations Views Reference, SC34-6463
CICSPlex SM Monitor Views Reference, SC34-6464
CICSPlex SM Managing Workloads, SC34-6465
CICSPlex SM Managing Resource Usage, SC34-6466
CICSPlex SM Managing Business Applications, SC34-6467

Programming
CICSPlex SM Application Programming Guide, SC34-6468
CICSPlex SM Application Programming Reference, SC34-6469

Diagnosis
CICSPlex SM Resource Tables Reference, SC34-6470
CICSPlex SM Messages and Codes, GC34-6471
CICSPlex SM Problem Determination, GC34-6472

CICS family books
Communication
CICS Family: Interproduct Communication, SC34-6473
CICS Family: Communicating from CICS on System/390, SC34-6474

Licensed publications

The following licensed publications are not included in the unlicensed version of the Information Center:
CICS Diagnosis Reference, LY33-6102
CICS Data Areas, LY33-6103
CICS Supplementary Data Areas, LY33-6104
CICS Debugging Tools Interfaces Reference, LY33-6105

Accessibility

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

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

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

IBM Personal Communications provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need in your CICS system.

Some accessibility features may not be available when using the application assembly tools for enterprise beans (ATK and AAT), which are components of WebSphere Application Server. You should consult the documentation that comes with WebSphere Application Server to determine which accessibility features are available when using these tools.

If you use the resource manager for enterprise beans to work with EJB resources, the accessibility features are those that your Web browser provides. In particular, note that the help that is presented when you allow the mouse pointer to hover over part of the display, is also available through the help function on that panel.

Index

Special characters

> 32K COMMAREAs (channels)
    DELETE CONTAINER (CHANNEL) command 226
    GET CONTAINER (CHANNEL) command 227
    MOVE CONTAINER (CHANNEL) command 229
    PUT CONTAINER (CHANNEL) command 231
    START CHANNEL command 233

A

abend codes 245
ABSTIME option
    CONVERTTIME command 153
ACTION option
    WEB CONVERSE command 100
    WEB SEND command (Client) 92
    WEB SEND command (Server) 121
ANALYZER attribute
    URIMAP definition 137
analyzer program 133
    chunked transfer-coding 113
analyzer programs 162
    DFHWBAAX 162
    DFHWBADX 162
ANALYZERSTAT option
    INQUIRE URIMAP command 156
API (application programming interface)
    JCICS changes 239
    modified API commands 236
    new API commands 225
API changes
    CICSPlex SM 304, 307
API command
    EXTRACT STATISTICS 315
application programming interface (API)
    JCICS changes 239
    modified API commands 236
    new API commands 225
application programming interface, CICSPlex SM
    changed resource tables
        CICSRGN 190, 269, 347
        CONNECT 252
        DOCDEF 170
        DOCTEMP 170
        EJCODEF 190
        EJCOSE 190
        PROGDEF 269, 347
        PROGRAM 269, 347
        TASK 170, 190, 252, 269
        TCPDEF 170, 190
        TCPIPGBL 190
        TCPIPS 190
        TERMDEF 341
        WORKREQ 78
    new resource tables
        HOST 170
        PIPEDEF 78

application programming interface, CICSPlex SM
  (continued)
    new resource tables *(continued)*
        PIPELINE 78
        URIMAP 170
        URIMPDEF 170
        URIMPGBL 170
        WEBSERV 78
        WEBSVDEF 78
AS option
    MOVE CONTAINER (CHANNEL) command 230
assistant, Web services 36

B

BAS 293
batch utility
    Web services assistant 36
BATCHREP resource table 307
benefits 197
big COMMAREAs 197, 198, 202, 218, 225
big COMMAREAs (channels)
    DELETE CONTAINER (CHANNEL) command 226
big COMMAREAs, channels 226, 227, 229, 231, 233
BINDING option
    INQUIRE WEBSERVICE command 68
BTS activities 216
business application services 293

C

CCRL transaction 182
CERTIFICATE attribute
    URIMAP definition 138
CERTIFICATE option
    INQUIRE URIMAP command 156
    WEB OPEN command 87
certificate revocation list 182
changes to CICS externals
    abend codes 245
    API 225
    CICS sample programs 244
    messages 245
    monitoring 242
    problem determination 245
    statistics 244
    trace points 248
    user-replaceable programs 240
channel commands
    DELETE CONTAINER (CHANNEL) 226
    GET CONTAINER (CHANNEL) 227
    MOVE CONTAINER (CHANNEL) 229
    PUT CONTAINER (CHANNEL) 231
    START CHANNEL 233
CHANNEL option
    ASSIGN command 236
    DELETE CONTAINER (CHANNEL) command 226

X

Notices

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

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

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

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

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

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

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

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

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Sending your comments to IBM

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

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

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

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

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

You can send your comments to IBM in any of the following ways:
- By mail, to this address:

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

Whichever you use, ensure that you include:
- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

397

IBM ®

Product Number: 5655-M15

Spine information:

IBM

CICS Transaction Server for z/OS

Release Guide

Version 3
Release 1