CICS Transaction Server for z/OS
Version 4 Release 1

Internet Guide

SC34-7021-03

CICS Transaction Server for z/OS
Version 4 Release 1

Internet Guide

SC34-7021-03

Note
Before using this information and the product it supports, read the information in "Notices" on page 437.

This edition applies to Version 4 Release 1 of CICS Transaction Server for z/OS (product number 5655-S97) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

What this book is about

This manual documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Version 4 Release 1.

This manual explains how to set up and manage CICS® Web support to enable CICS regions to act as HTTP servers and HTTP clients, and how to write CICS application programs that interact with Web clients and servers.

What you need to know to understand this book

This book assumes that you are familiar with CICS, either as a system administrator or as a system or application programmer.

Changes in CICS Transaction Server for z/OS, Version 4 Release 1

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

- *CICS Transaction Server for z/OS What's New*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.2*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.1*
- *CICS Transaction Server for z/OS Upgrading from CICS TS Version 2.3*

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

Part 1. Overview: CICS and HTTP

An overview of Internet, HTTP, TCP/IP and CICS Web support concepts, and information about the structure and behavior of CICS Web support.

Chapter 1. Connecting CICS to the Web

CICS can interface with the Web as a server, receiving requests from Web clients; or as a client, making requests to a server.

Web client requests serviced by CICS applications

Using CICS Web support

CICS Web support enables a CICS region to act as an HTTP server.

- CICS Web support can provide static responses to Web clients, using CICS documents or static files.
- Web-aware user application programs can receive and analyze HTTP requests, and provide dynamic application-generated responses.
- CICS Web support includes a range of CICS services supporting Web client access to non-Web-aware applications. Web clients can make requests to access CICS programs which are designed to communicate with virtual 3270 terminals, and CICS programs which are designed to be linked to from another CICS application using COMMAREAs or channels.
- CICS Web support also supports non-HTTP requests from clients.

Using Web services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface which is described in a machine-processable format (specifically, Web Services Definition Language - or WSDL). CICS Transaction Server Version 3 can be a requester or a provider of Web services.

Web services are described in the *CICS Web Services Guide*.

Using IBM® HTTP Server

IBM HTTP Server provides access to CICS applications through the External CICS Interface (EXCI) and the CICS business logic interface.

For more information, see Appendix K, “The CICS Web server plug-in,” on page 433 and Chapter 27, “Introduction to the CICS business logic interface,” on page 361.

Using CICS Transaction Gateway

The CICS Transaction Gateway provides a set of Web server facilities for access to CICS applications by a Web client. These include Java classes and Java beans for writing application-specific server programs (servlets) and browser programs (applets), as well as IBM-supplied code for common functions. There are classes for access to both traditional and object-oriented CICS applications. Applets and servlets use CICS-supplied classes to construct ECI (External Call Interface) and EPI (External Presentation Interface) requests. (Note that CICS Transaction Gateway for z/OS® supports ECI but not EPI.) For more information, see *CICS Transaction Gateway: z/OS Administration*.

For guidance about choosing a Web solution, see the IBM Redbooks publication *Revealed! Architecting Web Access to CICS*, SG24-5466, which is available from http://www.redbooks.ibm.com/redbooks/pdfs/sg245466.pdf.

CICS applications accessing the Web

CICS Web support enables a CICS region to act as an HTTP client. A user application program in CICS can initiate a request to an HTTP server, and receive responses from it. CICS Web support handles the messages in response to **EXEC CICS WEB** commands in the user application program.

Chapter 2. Internet, HTTP and TCP/IP concepts

This section explains key elements of the Hypertext Transfer Protocol (HTTP) and the Transmission Control Protocol/Internet Protocol (TCP/IP).

TCP/IP protocols

TCP/IP is a family of communication protocols used to connect computer systems in a network, and is named after two of the protocols in the family: Transmission Control Protocol (TCP) and Internet Protocol (IP). Hypertext Transfer Protocol (HTTP) is a member of the TCP/IP family.

The protocols within the TCP/IP family correspond, in many cases, with the layers of the Open Systems Interconnection (OSI) model. Table 1 shows HTTP and the underlying layers of the TCP/IP family in terms of the OSI model. The Systems Network Architecture (SNA) layers, which approximately match the OSI layers, are also shown.

Table 1. The layers of the TCP/IP protocol family

Layer	OSI	SNA	TCP/IP
7	Application	Application	HTTP
6	Presentation	Presentation	(empty)
5	Session	Data flow	(empty)
4	Transport	Transmission	TCP
3	Network	Path control	IP
2	Data link	Data link	Subnetwork
1	Physical	Physical	

Internet Protocol (IP)
IP is a network-layer protocol that provides a connectionless data transmission service that is used by TCP. Data is transmitted link by link; an end-to-end connection is never set up during the call. The unit of data transmission is the *datagram*.

Transmission Control Protocol (TCP)
TCP is a transport-layer protocol that provides a reliable, full duplex, connection-oriented data transmission service. Most Internet applications use TCP.

Hypertext Transfer Protocol (HTTP)
HTTP is an application-layer protocol that is used for distributed, collaborative, hypermedia information systems. HTTP is the protocol used between Web clients and Web servers.

Many TCP/IP implementations provide an application programming interface to the TCP protocol (that is, to the transport layer). This interface is commonly known as the *Sockets interface*. The TCP/IP Sockets interface for CICS is the z/OS Communications Server IP CICS Sockets interface. This is supplied with z/OS Communications Server, not with CICS, and is an integral part of z/OS. It is not

part of CICS Web support and does not use the CICS SO domain. *z/OS Communications Server: IP CICS Sockets Guide*, SC31-8807, describes the CICS Sockets interface.

IP addresses

Each server or client on a TCP/IP internet is identified by a numeric IP (Internet Protocol) address. There are two types of IP address, the IPv4 (IP version 4) address and the IPv6 (IP version 6) address.

IP addresses are managed and allocated to users by the Internet Assigned Numbers Authority (IANA) and its delegates. An internet is a collection of networks; therefore, the internet address specifies both the network and the individual host. This specification varies with the size of the network.

IPv6 addresses

IPv6 addresses are 128-bit addresses, usually expressed in hexadecimal notation:

```
IP address in hexadecimal notation :   '00010022033344440000000abc0def0'x
Halfword 0:  0001 hexadecimal
Halfword 1:  0022 hexadecimal
Halfword 2:  0333 hexadecimal
Halfword 3:  4444 hexadecimal
Halfword 4:  0000 hexadecimal
Halfword 5:  0000 hexadecimal
Halfword 6:  abc0 hexadecimal
Halfword 7:  def0 hexadecimal
IP address in colon hexadecimal notation: 1:22:333:4444::abc0:def0

IP address in hexadecimal notation :   '00000000000000000000ffff01020304'x
Halfword 0:  0000 hexadecimal
Halfword 1:  0000 hexadecimal
Halfword 2:  0000 hexadecimal
Halfword 3:  0000 hexadecimal
Halfword 4:  0000 hexadecimal
Halfword 5:  ffff hexadecimal
Halfword 6:  0102 hexadecimal
Halfword 7:  0304 hexadecimal
IP address in colon hexadecimal notation: ::ffff:1.2.3.4 or ::ffff:0102:0304
```

The address consists of eight halfword fields. Zeros are treated in the following ways in the address output:

- If a field contains leading zeros, these are ignored; for example, `0001` is represented as `1`
- If one or more consecutive fields in the address contain the value `0000`, these fields are expressed using the notation `::`

  For example, `0000000000000ffff` is represented as `::ffff`

  The `::` substitution is used once only in an address, to avoid confusion in calculating how many fields had been substituted.

IPv4 addresses

IPv4 addresses are 32-bit addresses, usually expressed in dotted decimal notation:

```
IP address in hexadecimal notation :   '817EB263'x
Byte 0:  81 hexadecimal = 129 decimal
Byte 1:  7E hexadecimal = 126 decimal
Byte 2:  B2 hexadecimal = 178 decimal
Byte 3:  63 hexadecimal =  99 decimal
IP address in dotted decimal notation: 129.126.178.99
```

In this example, 129.126 specifies the network and 178.99 specifies the host on that network.

IP address formats

CICS accepts IPv4 and IPv6 addresses in specific formats for processing.

IPv6 address formats

CICS accepts IPv6 addresses in the following format only:

- As a native IPv6 colon hexadecimal address without square brackets or /nn notation; for example, ::a:b:c:d

IPv6 address syntax is described in more detail in RFC 4291, *IP Version 6 Addressing Architecture*, available at http://www.ietf.org/rfc/rfc4291.txt.

IPv4 address formats

CICS accepts IPv4 addresses in the following formats:

- A native IPv4 dotted decimal address without /nn notation; for example, 1.2.3.4
- An IPv4 address that has been migrated to IPv6 format (an IPv4-mapped IPv6 address); for example, ::ffff:1.2.3.4
  - Internally, CICS translates the address into the binary equivalent of 0:0:0:0:0:ffff:0102:0304
- An IPv6 compatible address (an IPv4-compatible IPv6 address); for example, ::1.2.3.4
  - Internally, CICS translates the address into the binary equivalent of 0:0:0:0:0:0:0102:0304

  This exception applies:
  - CICS does not allow the following entries:
    - 0.0.0.0
    - ::0.0.0.0
    - ::0

Irrespective of the format you have specified for your IPv4 address, CICS displays all IPv4 addresses as a native IPv4 dotted decimal address; for example, 1.2.3.4

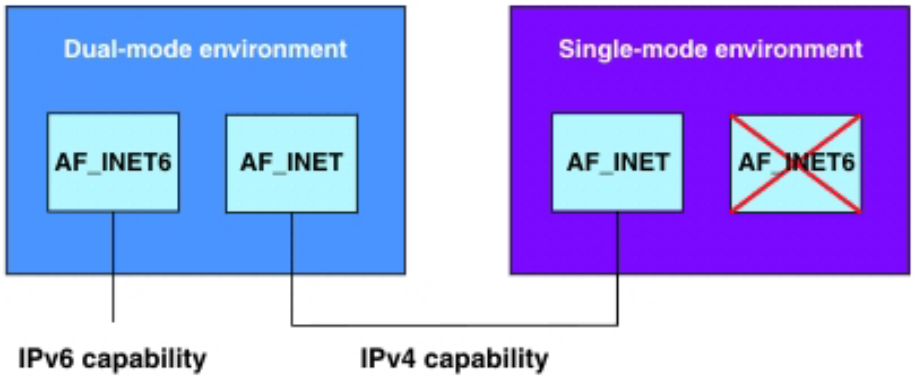
Understanding IPv6 and CICS

IPv6 is the protocol designed to replace IPv4. To use IPv6 addressing, the sending and receiving environments must support dual-mode addressing (IPv4 and IPv6) and your CICS regions must be running at the correct level of CICS.

Infrastructure requirements for IPv6

A dual-mode TCP/IP implementation is required to allow both IPv4 and IPv6 addressing. A single-mode (IPv4) environment uses the AF_INET address family when it establishes a connection between an AF_INET socket and another AF_INET socket in another region. IPv6 addresses are not supported over AF_INET sockets; these addresses require the AF_INET6 address family and AF_INET6 sockets in the sending and receiving regions to establish a connection. Dual-mode environments provide both AF_INET and AF_INET6 sockets. For more information on AF_INET and AF_INET6, see the *z/OS Communications Server IPv6 Network and Application Design Guide*.

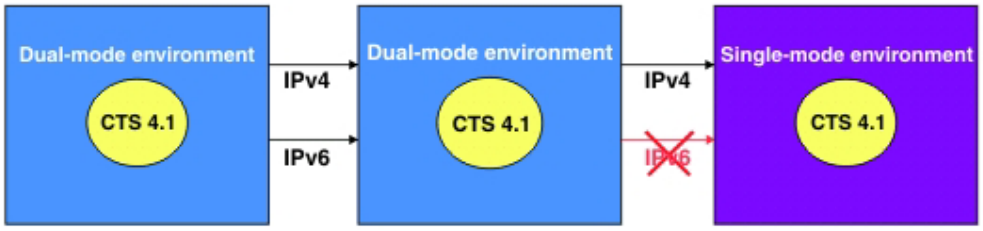
The figure shows that a single-mode environment does not have IPv6 capability, because it does not have an AF_INET6 socket.



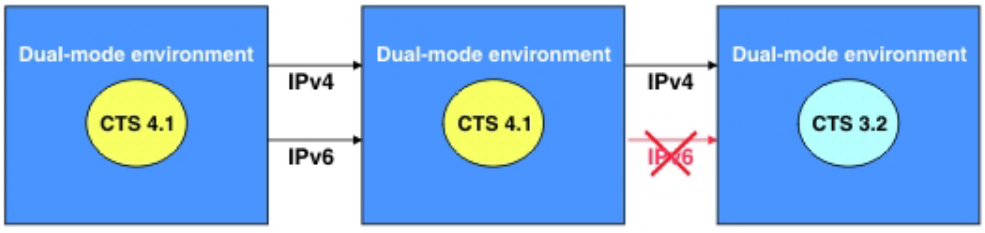
CICS requirements for IPv6

You need a minimum level of CICS TS 4.1 to communicate using IPv6. The CICS TS 4.1 region must be running in a dual-mode (IPv4 and IPv6) environment and the client or server that CICS is communicating with must also be running in a dual-mode environment.

The figure shows CICS-to-CICS communication, where two dual-mode CICS TS 4.1 environments can communicate using either IPv4 or IPv6 addressing. A single-mode CICS TS 4.1 environment is also connected, but can communicate using IPv4 only.



The figure shows CICS-to-CICS communication, where two dual-mode CICS TS 4.1 environments can communicate using either IPv4 or IPv6 addressing. A dual-mode CICS TS 3.2 environment is also connected, but can communicate using IPv4 only.



Host names

A host, or Web site, on the Internet is identified by a host name, such as `www.example.com`. Host names are sometimes called domain names. Host names are mapped to IP addresses, but there is not a one-to-one relationship between a host name and an IP address.

A host name is used when a Web client makes an HTTP request to a host. It is possible for the user making the request to specify the IP address of the server rather than the host name, but that is now unusual on the Internet. Host names are more convenient for users than numeric IP addresses. Companies, organizations and individuals frequently choose host names for their Web sites that can be easily remembered by users.

More importantly in modern HTTP implementations, the use of host names in HTTP requests means that:

- Services in the name of one host can be provided by many servers, which have different IP addresses.
- One server, with one IP address, can provide services in the name of many hosts. This is known as **virtual hosting**. "Virtual hosting" explains this process.

Host names are mapped to IP addresses by a server known as a **DNS server**, or **domain name server**. DNS stands for Domain Name Service. In a large network, many DNS servers may collaborate to provide the mapping between host names and IP addresses.

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.

Port numbers

Within a server, it is possible for more than one user process to use TCP at the same time. To identify the data associated with each process, port numbers are used. Port numbers are 16-bit, and numbers up to 65535 are possible, although in practice only a small subset of these numbers are commonly used.

When a client process first contacts a server process, it may use a *well-known port number* to initiate communication. Well-known port numbers are assigned to particular services throughout the Internet, by IANA, the Internet Assigned Numbers Authority. The well-known port numbers are in the range 0 through 1023. Some examples are shown in Table 2:

Table 2. Services and their well-known port numbers

Service	Well-known port number
File Transfer Protocol (FTP)	21
Telnet	23
Hypertext Transfer Protocol (HTTP)	80
HTTP with Secure Sockets Layer (SSL)	443
CORBA Internet Inter-ORB Protocol (IIOP)	683
CORBA IIOP with SSL	684

The CICS External Call Interface (ECI) has a registered port number, 1435.

Well-known ports are used only to establish communication between client and server processes. When this has been done, the server allocates an *ephemeral port number* for subsequent use. Ephemeral port numbers are unique port numbers which are assigned dynamically when processes start communicating. They are released when communication is complete.

IANA media types and character sets

The Internet Assigned Numbers Authority (IANA) is the international body responsible for assigning names for protocols used on the Internet.

- IANA media types are names for the types of data that are commonly transmitted over the Internet. They are described at http://www.iana.org/assignments/media-types/

  Text media types (such as a type that begins with `text/`, or a type that contains `+xml`) are identified by RFC 3023, which is available at http://www.ietf.org/rfc/rfc3023.txt.
- IANA character sets are the names of character set registries. They are described at http://www.iana.org/assignments/character-sets

  CICS does not support all the IANA character sets for code page conversion. The character sets that CICS supports are described in Appendix A, "HTML coded character sets," on page 373.

The components of a URL

A URL (Uniform Resource Locator) is a specific type of URI (Universal Resource Identifier). A URL normally locates an existing resource on the Internet. A URL is used when a Web client makes a request to a server for a resource.

The concepts of the URI and the URL are defined by the Internet Society and IETF (Internet Engineering Task Force) Request for Comments document RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax* (http://www.ietf.org/rfc/rfc2396.txt). Briefly, a URI is defined as any character string that identifies a resource. A URL is defined as those URIs that identify a resource by its location or by the means used to access it, rather than by a name or other attribute of the resource.

A newer form of resource identifier, the IRI (Internationalized Resource Identifier), permits the use of characters and formats that are suitable for national languages other than English. An IRI can be used in place of a URI or URL where the applications involved with the request and response support this. For more information about IRIs, see “Internationalized Resource Identifiers (IRIs)” on page 249.

A URL for HTTP (or HTTPS) is normally made up of three or four components:

1. **A scheme.** The scheme identifies the protocol to be used to access the resource on the Internet. It can be HTTP (without SSL) or HTTPS (with SSL).
2. **A host.** The host name identifies the host that holds the resource. For example, `www.example.com`. A server provides services in the name of the host, but there is not a one-to-one mapping between hosts and servers. “Host names” on page 9 explains more about host names.

   Host names can also be followed by a **port number**. “Port numbers” on page 9 explains more about these. Well-known port numbers for a service are normally omitted from the URL. Most servers use the well-known port numbers for HTTP and HTTPS , so most HTTP URLs omit the port number.
3. **A path.** The path identifies the specific resource within the host that the Web client wants to access. For example, `/software/htp/cics/index.html`.
4. **A query string.** If a query string is used, it follows the path component, and provides a string of information that the resource can use for some purpose (for example, as parameters for a search or as data to be processed). The query string is usually a string of name and value pairs, for example, `term=bluebird`. Name and value pairs are separated from each other by an ampersand (&), for example, `term=bluebird&source=browser-search`.

The scheme and host components of a URL are not defined as case-sensitive, but the path and query string are case-sensitive. Usually, the whole URL is specified in lower case.

The components of the URL are combined and delimited as follows:

```
scheme://host:port/path?query
```

- The scheme is followed by a colon and two forward slashes.
- If a port number is specified, that number follows the host name, separated by a colon.
- The path name begins with a single forward slash.
- If a query string is specified, it is preceded by a question mark.

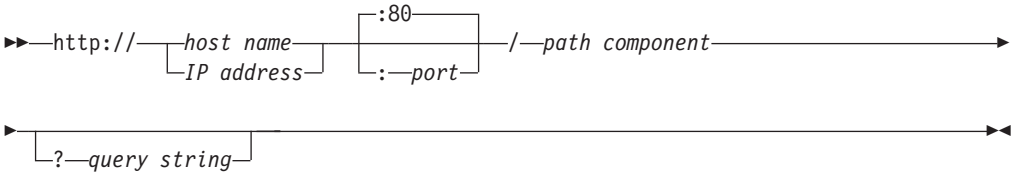


Figure 1. Syntax of an HTTP URL

This is an example of an HTTP URL:

```
http://www.example.com/software/index.html
```

If a port number was specified, the URL would be:

```
http://www.example.com:1030/software/index.html
```

A URL can be followed by a fragment identifier. The separator used between the URL and the fragment identifier is the # character. A fragment identifier is used to point a Web browser to a reference or function within the item that it has just retrieved. For example, if the URL identifies an HTML page, a fragment identifier can be used to indicate a subsection within the page, using the ID of the subsection. In this case, the Web browser normally displays the page to the user so that the subsection is visible. The action taken by the Web browser for a fragment identifier differs depending on the media type of the item and the defined meaning of the fragment identifier for that media type.

Other protocols, such as File Transfer Protocol (FTP) or Gopher, also use URLs. The URLs used by these protocols may have a different syntax to the one used for HTTP.

The HTTP protocol

The correct format for HTTP requests and responses depends on the version of the HTTP protocol (or HTTP specification) that is used by the client and by the server.

The versions of the HTTP protocol (or "HTTP versions") commonly used on the Internet are HTTP/1.0, which is an earlier protocol including fewer functions, and HTTP/1.1, which is a later protocol including more functions. The client and server might use different versions of the HTTP protocol. Both client and server must state the HTTP version of their request or response in the first line of their message.

Internet Society and IETF (Internet Engineering Task Force) Request for Comments documents (known as RFCs) provide the official definitions for the HTTP protocol. These documents are:

HTTP/1.0
RFC 1945, *Hypertext Transfer Protocol - HTTP/1.0*, available from http://www.ietf.org/rfc/rfc1945.txt

HTTP/1.1
RFC 2616, *Hypertext Transfer Protocol - HTTP/1.1*, available from http://www.ietf.org/rfc/rfc2616.txt

The RFCs state the actions that a client and a server should perform to exchange requests and responses in an appropriate way for each version of the HTTP protocol. These actions are described as "requirements". A client or server that fulfils the requirements for its version of the HTTP protocol is said to be "compliant" with the HTTP specification.

HTTP requests

An HTTP request is made by a client, to a named host, which is located on a server. The aim of the request is to access a resource on the server.

To make the request, the client uses components of a URL (Uniform Resource Locator), which includes the information needed to access the resource. “The components of a URL” on page 10 explains URLs.

A correctly composed HTTP request contains the following elements:

1. A request line.

2. A series of HTTP headers, or header fields.
3. A message body, if needed.

Each HTTP header is followed by a carriage return line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used (to give an empty line), and then any message body begins.

Request line

The request line is the first line in the request message. It consists of at least three items:

1. A **method**. The method is a one-word command that tells the server what it should do with the resource. For example, the server could be asked to send the resource to the client.
2. The path component of the URL for the request. The path identifies the resource on the server.
3. The HTTP version number, showing the HTTP specification to which the client has tried to make the message comply.

An example of a request line is:

```
GET /software/htp/cics/index.html HTTP/1.1
```

In this example:

- the method is `GET`
- the path is `/software/htp/cics/index.html`
- the HTTP version is `HTTP/1.1`

A request line might contain some additional items:

- A query string. This provides a string of information that the resource can use for some purpose. It follows the path, and is preceded by a question mark.
- The scheme and host components of the URL, in addition to the path. When the resource location is specified in this way, it is known as the **absolute URI** form. For HTTP/1.1, this form is used when a request will go through a proxy server. Also for HTTP/1.1, if the host component of the URL is not included in the request line, it must be included in the message in a Host header.

HTTP headers

HTTP headers are written on a message to provide the recipient with information about the message, the sender, and the way in which the sender wants to communicate with the recipient. Each HTTP header is made up of a name and a value. The HTTP protocol specifications define the standard set of HTTP headers, and describe how to use them correctly. HTTP messages can also include extension headers, which are not part of the HTTP/1.1 or HTTP/1.0 specifications.

The HTTP headers for a client's request contain information that a server can use to decide how to respond to the request. For example, the following series of headers can be used to specify that the end user only wants to read the requested document in French or German, and that the document should only be sent if it has changed since the date and time when the client last obtained it:

```
Accept-Language: fr, de
If-Modified-Since: Fri, 10 Dec 2004 11:22:13 GMT
```

An empty line (that is, a CRLF alone) is placed in the request message after the series of HTTP headers, to divide the headers from the message body.

Message body

The body content of any HTTP message can be referred to as a message body or **entity body**. Technically, the entity body is the actual content of the message. The message body contains the entity body, which can be in its original state, or can be encoded in some way for transport, such as by being broken into chunks (chunked transfer-coding). The message body of a request may be referred to for convenience as a request body.

Message bodies are appropriate for some request methods and inappropriate for others. For example, a request with the POST method, which sends input data to the server, has a message body containing the data. A request with the GET method, which asks the server to send a resource, does not have a message body.

HTTP responses

An HTTP response is made by a server to a client. The aim of the response is to provide the client with the resource it requested, or inform the client that the action it requested has been carried out; or else to inform the client that an error occurred in processing its request.

An HTTP response contains:

1. A status line.
2. A series of HTTP headers, or header fields.
3. A message body, which is usually needed.

As in a request message, each HTTP header is followed by a carriage return line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used (to give an empty line), and then the message body begins.

Status line

The status line is the first line in the response message. It consists of three items:

1. The HTTP version number, showing the HTTP specification to which the server has tried to make the message comply.
2. A **status code**, which is a three-digit number indicating the result of the request.
3. A **reason phrase**, also known as status text, which is human-readable text that summarizes the meaning of the status code.

An example of a response line is:

```
HTTP/1.1 200 OK
```

In this example:

- the HTTP version is `HTTP/1.1`
- the status code is `200`
- the reason phrase is `OK`

"Status codes and reason phrases" on page 15 explains more about these elements of the status line.

HTTP headers

The HTTP headers for a server's response contain information that a client can use to find out more about the response, and about the server that sent it. This information can assist the client with displaying the response to a user, with storing (or caching) the response for future use, and with making further requests to the server now or in the future. For example, the following series of headers tell the client when the response was sent, that it was sent by CICS, and that it is a JPEG image:

```
Date: Thu, 09 Dec 2004 12:07:48 GMT
Server: IBM_CICS_Transaction_Server/3.1.0(zOS)
Content-type: image/jpg
```

In the case of an unsuccessful request, headers can be used to tell the client what it must do to complete its request successfully.

An empty line (that is, a CRLF alone) is placed in the response message after the series of HTTP headers, to divide the headers from the message body.

Message body

The message body of a response may be referred to for convenience as a response body.

Message bodies are used for most responses. The exceptions are where a server is responding to a client request that used the HEAD method (which asks for the headers but not the body of the response), and where a server is using certain status codes.

For a response to a successful request, the message body contains either the resource requested by the client, or some information about the status of the action requested by the client. For a response to an unsuccessful request, the message body might provide further information about the reasons for the error, or about some action the client needs to take to complete the request successfully.

Status codes and reason phrases

In the HTTP response that is sent to a client, the status code, which is a three-digit number, is accompanied by a reason phrase (also known as status text) that summarizes the meaning of the code. Along with the HTTP version of the response, these items are placed in the first line of the response, which is therefore known as the status line.

The status codes are classified by number range, with each class of codes having the same basic meaning.

- The range 100-199 is classed as Informational.
- 200-299 is Successful.
- 300-399 is Redirection.
- 400-499 is Client error.
- 500-599 is Server error.

When describing a range as a whole, it may be named as "1xx", "2xx", and so on. The HTTP protocol specifications do not define any status codes of 600 or greater.

Only a few status codes in each range are defined by the HTTP/1.0 and HTTP/1.1 specifications. The HTTP/1.1 specification includes more status codes than the HTTP/1.0 specification.

The reason phrases defined in the HTTP specifications (for example, "Not Found" or "Bad Request") are recommended but optional. The HTTP/1.1 specification says that the reason phrases for each status code may be replaced by local equivalents.

The 200 (OK) status code is used for a normal response that provides the full resource requested by the Web client. Most other status codes are used in situations where there is an error that prevents fulfilment of the request, or where the client needs to do something else in order to complete its request successfully, such as following a redirection URL, or amending the request so that it is acceptable to the server.

The HTTP headers for the response, or the response body, or both, may provide further instructions and information for the client. The HTTP specifications include requirements and suggestions for the content of responses with each status code. The requirements specify:

- Any HTTP headers that must, or may, be used on the response. For example, if you use the status code 405 (Method not allowed), you must use the Allow header to state the methods which *are* allowed.
- Whether or not a response body should be used. For example, message bodies are not allowed with status codes 204, 205, and 304.
- If a response body is used, what information it can provide. For example, message bodies for a redirection can provide a hyperlink for the redirection URL.

For full information about the meaning and correct use of status codes, you should consult the HTTP specification to which you are working. See “The HTTP protocol” on page 12 for more information about the HTTP specifications.

Escaped and unescaped data

To assist with the correct transmission and interpretation of an HTTP request, there are restrictions on the use of certain characters within a URL. These characters must be converted to a safe format when the request is transmitted.

In a URI or a URL, characters that have a special purpose in the context of one or more URI or URL components are known as **reserved characters**. For example, the characters `/` , `?`, `&` and `:` are used as delimiters for various components. Machine interpreters might misinterpret the URI or URL if the reserved characters are used for any reason other than their special purpose.

Also, certain characters are disallowed, or **excluded**, from use anywhere in a URI or URL, either because they are a potential cause of confusion for machine or human users, or because they are known to cause problems for some machine interpreters. For example, the space character is not permitted in a URL.

The Internet Society and IETF (Internet Engineering Task Force) Request for Comments document RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, lists the characters that are reserved or excluded in URIs and URLs. RFC 2396 is available from http://www.ietf.org/rfc/rfc2396.txt.

If reserved characters are wanted in a URL for any reason other than their special purpose, or if excluded characters are wanted in a URL, they must be **escaped** when a request containing components of the URL is sent to a server. This includes characters in data that is sent in a query string.

Characters are escaped by being replaced with a three character string of the form `%xx` where `xx` is the ASCII hexadecimal representation of the reserved character. For example, the / character is replaced by `%2F`. As a special case, the space character may be replaced by `+`. Because of this format, escaping is also known as **percent-encoding**.

When the request reaches the server, the server can **unescape** the escaped characters, that is, convert them from the escape sequence back to the original character. Unescaping should only take place after the information in the URL and query string has been parsed, to avoid the risk of the parsing application misinterpreting the reserved or excluded characters.

Form data in a request, whether it is presented in the URL or in the message body, is normally sent with special characters escaped, because the default encoding for forms (**application/x-www-form-urlencoded**) escapes reserved or excluded characters. "HTML forms" explains more about this.

HTML forms

In HTML, forms are areas delimited by a `<form>` tag, containing text input boxes, buttons, check boxes, and other features of a graphical user interface. Forms are used by Web applications to allow end users to provide data to be sent to the server.

Within a form, the elements with which users can interact to provide data are known as **form fields**. Each form field is given a name in the HTML, which identifies it to the server application, but is not visible to the user.

Although the various elements of a form appear different to the user, they all transmit information to the server application in the same way: as a series of name and value pairs, separated by `&` characters. Each name is the name of a form field, and the value is the data produced by the user's actions. For example, if a form contained two text input boxes for a user to enter their first and last name, the data might look like this:

```
firstname=Maria&lastname=Smith
```

The form data is transmitted to the server in one of two ways, depending on which method (GET or POST) is specified in the `<form>` tag:

- When the method is GET, the form data is transmitted in a query string in the URL.
- When the method is POST, the form data is transmitted in the message body.

The character set that is required for encoding the form data is specified by the CHARACTERSET option, and should match the forms encoding determined by the corresponding HTML form (see "How the client encoding is determined" on page 18 for more information).

Form data is normally transmitted with special characters escaped. "Escaped and unescaped data" on page 16 explains the purpose of escaping.

If the form is defined with the GET method, because the data is sent as a query string in the URL, reserved or excluded characters must always be escaped.

If the form is defined with the POST method, the data is sent in the message body. However, as defined in the HTML 2.0 specification, the default encoding type for all forms is **application/x-www-form-urlencoded**. (See http://www.w3.org/MarkUp/html-spec/html-spec_8.html#SEC8.2.1) When this encoding is used for a form with the POST method, although the data is sent in the message body, reserved or excluded characters are escaped, as they would be if they were in a URL.

If the alternative encoding type **multipart/form-data** is specified for the form (which is done using the ENCTYPE attribute on the HTML `<form>` tag), non-ASCII characters in field names should be escaped, but non-ASCII characters in field values do not need to be escaped. The data is also presented in a series of individual sections in the message body. Older applications might not support this encoding. CICS does support it. The **multipart/form-data** encoding is described in the Internet Society and IETF Request for Comments document RFC 1867, *Form-based File Upload in HTML* (http://www.ietf.org/rfc/rfc1867.txt).

How the client encoding is determined

The character encoding (**`charset`** parameter) used by HTTP clients for forms data (both for the GET and POST methods) is determined by information in the HTML form.

The HTTP client normally submits forms data using the same character encoding that was used for the HTML form, specified either by the **`charset`** parameter on the Content-Type header or using an equivalent `META` tag embedded in the HTML, for example:

```
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

The accept-charset attribute on the HTML FORM element can also be used to specify an additional acceptable character encoding. If you do not specify the code page, CICS obtains this information from the **`charset`** parameter. The HTML form character encoding is normally either ISO-8859-1 (CCSID 819) or UTF-8 (CCSID 1208), but is not restricted to these values.

The character encoding information is not normally present as part of the submitted form request, so if the default character set for the internet (ISO-8859-1) is not used, the application reading the form must specify the encoding using the CHARACTERSET keyword. If CHARACTERSET is omitted, but the HTTP client provides a charset value in a Content-Type header (this is not standard practice for HTML forms submission), then the charset value is used, otherwise CICS assumes ISO-8859-1.

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).

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. See “The HTTP protocol” on page 12 for more information about the HTTP specifications. Briefly, most request methods are idempotent if they are used on their own, because the same result is obtained each time the method is used. (The exception is the POST method, because it changes the resource on the server.) However, when a sequence of requests is issued during pipelining, the sequence might be non-idempotent, particularly if resources are being changed.

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.

Related tasks:

“Sample programs: pipelining requests to an HTTP server” on page 158
Sample programs DFH$WBPA (Assembler), DFH$WBPC (C), and DFH0WBPO (COBOL) demonstrate how CICS can pipeline client requests to an HTTP server.

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.

HTTP basic authentication

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and password) from a client. The client passes the authentication information to the server in an Authorization header. The authentication information is in base-64 encoding.

If a client makes a request for which the server expects authentication information, the server sends an HTTP response with a 401 status code, a reason phrase indicating an authentication error, and a WWW-Authenticate header. Most Web clients handle this response by requesting a user ID and password from the end user.

The format of a WWW-Authenticate header for HTTP basic authentication is:

```
WWW-Authenticate: Basic realm="Our Site"
```

The WWW-Authenticate header contains a realm attribute, which identifies the set of resources to which the authentication information requested (that is, the user ID and password) will apply. Web clients display this string to the end user when they request a user ID and password. Each realm may require different authentication information. Web clients may store the authentication information for each realm so that end users do not need to retype the information for every request.

When the Web client has obtained a user ID and password, it re-sends the original request with an Authorization header. Alternatively, the client may send the Authorization header when it makes its original request, and this might be accepted by the server, avoiding the challenge and response process.

The format of the Authorization header is:

```
Authorization: Basic userid:password
```

The user ID and password are encoded using the base-64 encoding scheme.

RFC 2617, *HTTP Authentication: Basic and Digest Access Authentication*, available from http://www.ietf.org/rfc/rfc2617.txt, has more detailed information about basic authentication.

Note: The HTTP basic authentication scheme can only be considered a secure means of authentication when the connection between the Web client and the server is secure. If the connection is insecure, the scheme does not provide sufficient security to prevent unauthorized users from discovering and using the authentication information for a server. If there is a possibility of a password being intercepted, basic authentication should be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information.

Chapter 3. CICS Web support concepts and structure

CICS Web support is a collection of CICS services that enable a CICS region to act both as an HTTP server, and as an HTTP client.

CICS as an HTTP server

When CICS is an HTTP server, a Web client can send an HTTP request to CICS and receive a response. The response can be a static response created by CICS from a document template or static file, or an application-generated response created dynamically by a user application program.

The actions of CICS as an HTTP server are controlled by:

1. System initialization parameters and resource definitions, including TCPIPSERVICE definitions and URIMAP definitions, which are used to configure CICS Web support and instruct CICS how to process requests and responses.
2. CICS utility programs, which can be used to analyze and process the HTTP requests and responses.
3. User-written application programs, which are used to receive the HTTP requests and provide material for HTTP responses. These can be Web-aware application programs designed for use with CICS Web support, or non-Web-aware CICS application programs that were not originally designed for use with CICS Web support.

The behavior of CICS Web support as an HTTP server is conditionally compliant with the HTTP/1.1 specification, as described in RFC 2616. See "The HTTP protocol" on page 12 for more information about the HTTP specifications.

CICS as an HTTP client

When CICS is an HTTP client, a user application program in CICS can initiate a request to an HTTP server, and receive a response from it.

The actions of CICS as an HTTP client are controlled by user-written application programs. The **EXEC CICS** WEB application programming interface includes commands that an application program can use to construct and initiate HTTP requests from CICS, and to receive responses sent by servers. URIMAP resource definitions can be used to provide information such as a URL or a client certificate label.

CICS Web support and non-HTTP messages

CICS Web support also supports non-HTTP requests from clients. You can use many of the components of CICS Web support, including TCPIPSERVICE definitions, CICS utility programs, and user-written application programs, to provide request handling for any request format that you have defined. Non-HTTP messages that are handled by CICS Web support use a special protocol (the USER protocol) on the TCPIPSERVICE resource definition, so that they are not subjected to the checks that CICS carries out for HTTP messages.

In CICS Transaction Server for z/OS, Version 4 Release 1, this facility is primarily intended to provide support for requests from user-written clients that use nonstandard request formats. The processing that takes place for requests is defined by the user. The facility does not provide specific support for any formally defined protocols which are used for client-server communication.

The support that CICS Web support provides for non-HTTP messages is not the same thing as the TCP/IP Sockets interface for CICS. The IP CICS Sockets interface supplied with z/OS Communications Server has an application programming interface which allows clients to communicate directly with CICS application programs over TCP/IP. CICS Web support is not involved with this process. *z/OS Communications Server: IP CICS Sockets Guide*, SC31-8807, describes the CICS Sockets interface.

Components of CICS Web support

CICS Web support includes some base components that are used for all CICS Web support tasks, and some task-specific components which you select and configure for individual CICS Web support tasks.

Base components

- **TCP/IP support** in CICS is provided by the CICS SO (sockets) domain, with network services (z/OS Communications Server and access to a DNS server) supplied by z/OS.
- **z/OS UNIX Systems Services** are used as part of TCP/IP support, and the CICS region needs to access these.
- **Secure Sockets Layer (SSL) support** is used to provide security for the CICS Web support implementation. CICS supports the Secure Sockets Layer (SSL) 3.0 protocol, and the Transport Layer Security (TLS) 1.0 protocol. (SSL 2.0 is not supported). Note that where the term SSL is used in CICS documentation, it normally refers to both SSL and TLS. The *CICS RACF Security Guide* has more information about SSL and TLS.
- **DOCCODEPAGE system initialization parameter** specifies the default host code page that is used by CICS document template support.
- **LOCALCCSID system initialization parameter** specifies the coded character set identifier for the local CICS region (which is the code page that CICS considers as the default for application programs).
- **TCPIP system initialization parameter** activates CICS TCP/IP services at startup.
- **WEBDELAY system initialization parameter** defines a timeout period for inactive CICS Web tasks, only where the Web 3270 bridge facility is involved. Timeout for other CICS Web tasks is handled by the RTIMOUT value for the relevant transaction, or (for CICS as an HTTP server) by the SOCKETCLOSE attribute on the TCPIPSERVICE definition.
- **The Sockets listener task (CSOL)** detects inbound TCP/IP connection requests, and invokes CICS Web support by attaching the Web attach task.
- **Web attach tasks (CWXN, CWXU or an alias)** receive data from the Web client and deal with initial processing of requests, including URIMAP matching, code page conversion of the HTTP headers, analysis of the request, and code page conversion of the message body. The tasks also pre-process chunked and pipelined messages received from a Web client. If a static response is delivered (using a URIMAP definition), the Web attach task handles this processing as well.

Resource definitions

- **TCPIPSERVICE resource definitions** are used to define each port that you use for CICS as an HTTP server, including security options for connections on that port, and timeout and maximum size limits for inbound requests. They are not used for CICS as an HTTP client.

  Note: The TCPIPSERVICE resource definitions are for use only with the CICS-provided TCP/IP services, and have nothing to do with the z/OS Communications Server IP CICS Sockets interface. The TCP/IP Socket Interface for CICS is supplied with z/OS Communications Server, which is an integral part of z/OS and does not use the CICS SO domain.
- **URIMAP resource definitions** match the URLs of requests from Web clients, or requests to an HTTP server, and provide CICS with information on how to process the requests. URIMAP definitions incorporate, and can replace, the CICS Web support processing functions that were provided before CICS Transaction Server for z/OS, Version 3 Release 1 by the analyzer program associated with the TCPIPSERVICE definition. URIMAP definitions can also be used to deliver a static response to a request from a Web client, without involving an application program.
- **TRANSACTION resource definitions** are used to define alias transactions for HTTP request processing. CICS supplies a resource definition for a default alias transaction, CWBA. When the Web attach task has completed initial processing for the request, if an application-generated response is to be produced, an alias transaction handles the remaining stages of processing. These include receiving the request, executing the application's business logic, construction of the HTTP response and code page conversion of the HTTP response.

User application programs

- **Web-aware application programs** can be designed for CICS Web support, using the EXEC CICS WEB and EXEC CICS DOCUMENT application programming interfaces. For CICS as an HTTP server, these programs can receive and analyze HTTP requests and provide application-generated responses to the Web client. For CICS as an HTTP client, a user application program in CICS can initiate an HTTP request to a server, and receive a response from it.
- **COMMAREA applications**, programs which are designed to be linked to from another program using a COMMAREA interface, can be accessed using CICS Web support with a converter program to convert their output into HTML for transmission to a Web client. Alternatively, you can write a Web-aware application program that links to a COMMAREA application and uses its output to provide HTTP responses.
- **3270 display applications**, programs which are designed to communicate with 3270 terminals, can be accessed using the Web Terminal Translation Application. The HTML output created by the Web Terminal Translation Application can be displayed in a Web browser.

Programming interfaces

- The **EXEC CICS WEB** application programming interface interprets and constructs HTTP requests and HTTP responses. Some commands are used for CICS as an HTTP server, some for CICS as an HTTP client, and some are for both forms of CICS Web support.
- The **EXEC CICS DOCUMENT** application programming interface constructs CICS documents to provide the body of a response or request that is sent out from CICS.

CICS Web support utility programs

- **Analyzer programs** are associated with TCPIPSERVICE definitions. They are used to interpret an HTTP request if a URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present. CICS supplies a default analyzer program DFHWBAAX, which provides basic error handling, and a sample analyzer program DFHWBADX, which supports requests using the URL format that CICS Web support used before CICS TS 3.1. Either of these analyzers can be used as a basis for your own analyzer program.
- **Converter programs** can be used to decode an HTTP request and construct input to a user application program. Web-aware application programs do not normally require converter programs, but they might be needed by non-Web-aware applications that were not designed for CICS Web support. CICS does not supply a converter program. You can write a number of converter programs and select any converter program in your CICS region to process a request.
- **Web error programs** provide an error response to the Web client when a request error or an abend occurs in the CICS Web support process. CICS supplies the Web error program DFHWBEP, which is used in most error situations, and the Web error application program DFHWBERX, which is used with the default analyzer DFHWBAAX when URIMAP matching fails (and can be specified for other situations). The Web error programs are user-replaceable, and they can be modified to customize or change the error response that is sent to the Web client in each error situation.
- **The Web Terminal Translation Application DFHWBTTA** (and its aliases for alternative processing, DFHWBTTB and DFHWBTTC) can be used to create HTML output from programs which are designed to communicate with 3270 terminals. The program uses the CICS 3270 bridge mechanism. Applications that do, and those that do not, use BMS are both supported. No application program changes are needed to use this feature.
- **The password expiry management program DFHWBPW** is used when basic authentication is specified for the connection, and the user's password has expired. The program takes the user through the process of setting a new password. You can customize or replace the Web pages presented to the user by DFHWBPW.

Document construction facilities

- **z/OS UNIX Systems Services files** can be served as the body of a response to an HTTP request from a Web client.
- **Document template support** enables message bodies to be built from fragments of HTML which are prepared offline.
- **BMS macros** construct HTML document templates from BMS map sets.

Code page conversion

CICS provides facilities to convert HTTP messages into a code page that is suitable for a user application program, or suitable for use on the Internet. CICS handles code page conversion using z/OS conversion services.

The code page conversion table (DFHCNV), which was required in earlier CICS releases, is not normally required for CICS Web support in CICS Transaction Server for z/OS, Version 4 Release 1. The exception is if you want to use an analyzer program that you coded in an earlier CICS release to reference DFHCNV. In this case, you must either continue to supply the code page conversion table, or make an update to the analyzer program. “Upgrading entries in the code page

conversion table (DFHCNV)" on page 55 has more information about this.

Task structure for CICS Web support

When CICS Web support is active in a CICS region, for CICS as an HTTP server, separate tasks are used to listen for inbound connection requests; to receive data from the socket and perform initial processing; and to cover work carried out by application programs in connection with a request. For CICS as an HTTP client, only one task applies, which is the task for the application program making the HTTP requests.

The Sockets listener task (CSOL)

This is a long running CICS task. There is one instance of the Sockets listener task in a CICS system.

The task detects inbound TCP/IP connection requests on all ports defined to CICS, and invokes the CICS service associated with the port. When the port is intended for CICS Web support (that is, HTTP or USER is specified as the protocol), the Web attach task is defined as the transaction in the TCPIPSERVICE resource definition for the port, so the listener attaches that task.

Web attach tasks (CWXN, CWXU or an alias)

When the TCPIPSERVICE definition for a port has the protocol HTTP, the default transaction ID for the Web attach task is CWXN. When the protocol is USER, the default is CWXU. An alias can be used instead, but the transaction always executes program DFHWBXN.

When a Web attach task is invoked by the Sockets listener task, the first thing it does is to issue a SOCKET RECEIVE request to receive data from the Web client. When some data has been received, the Web attach task deals with initial processing of the Web client's request.

- For an HTTP request (on the HTTP protocol), the initial processing includes URIMAP matching, code page conversion of the HTTP headers, analysis of the request, and code page conversion of the message body. The task also pre-processes chunked and pipelined messages received from a Web client. If an analyzer program is used, it is covered by this transaction.
- For a non-HTTP request (on the USER protocol), no initial processing takes place.

If a static response is delivered to an HTTP request (using a URIMAP definition), the Web attach task handles this processing as well. If an application-generated response is required, the Web attach task attaches an alias transaction.

There is an instance of the Web attach task for each individual request from a Web client which is in the initial stages of processing. 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 a request from the Web client has been passed to the alias 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, when a CWXN transaction remaining in the system would not be able to

attach alias transactions to process further requests.

Alias transactions for application-generated responses

When a Web attach task has completed initial processing for a request, if an application-generated response is to be produced, the Web attach task attaches the alias transaction which is specified for the remaining processing stages of that request. CICS supplies a resource definition for a default alias transaction, CWBA. Alias transactions are not used where a static response is provided.

An alias transaction handles the processing stages for an application-generated response, which include receiving the request, executing the application's business logic, constructing the HTTP response and code page conversion of the HTTP response. If a converter program is used to process the request, it is also handled by the alias transaction. There is an instance of an alias transaction for each HTTP request which is in these stages of processing.

CICS as an HTTP client

For CICS as an HTTP client, all activity caused by an application program that makes HTTP client requests is covered by a single task. This includes the application program's actions, the actions of CICS in sending requests and receiving responses, and socket activity. If the application program links to other programs using the **EXEC CICS** LINK command, these are also covered by the task. The task has the transaction ID that triggers the application program.

The task remains in the system from the beginning to the end of the application program's activity. The task may involve more than one request and response, and the application program may open and maintain more than one connection to a server. When the task ends, any open connections are automatically closed.

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.

Figure 2 on page 27 shows the processing that is carried out by CICS Web support to receive a request from a Web client and provide a response.

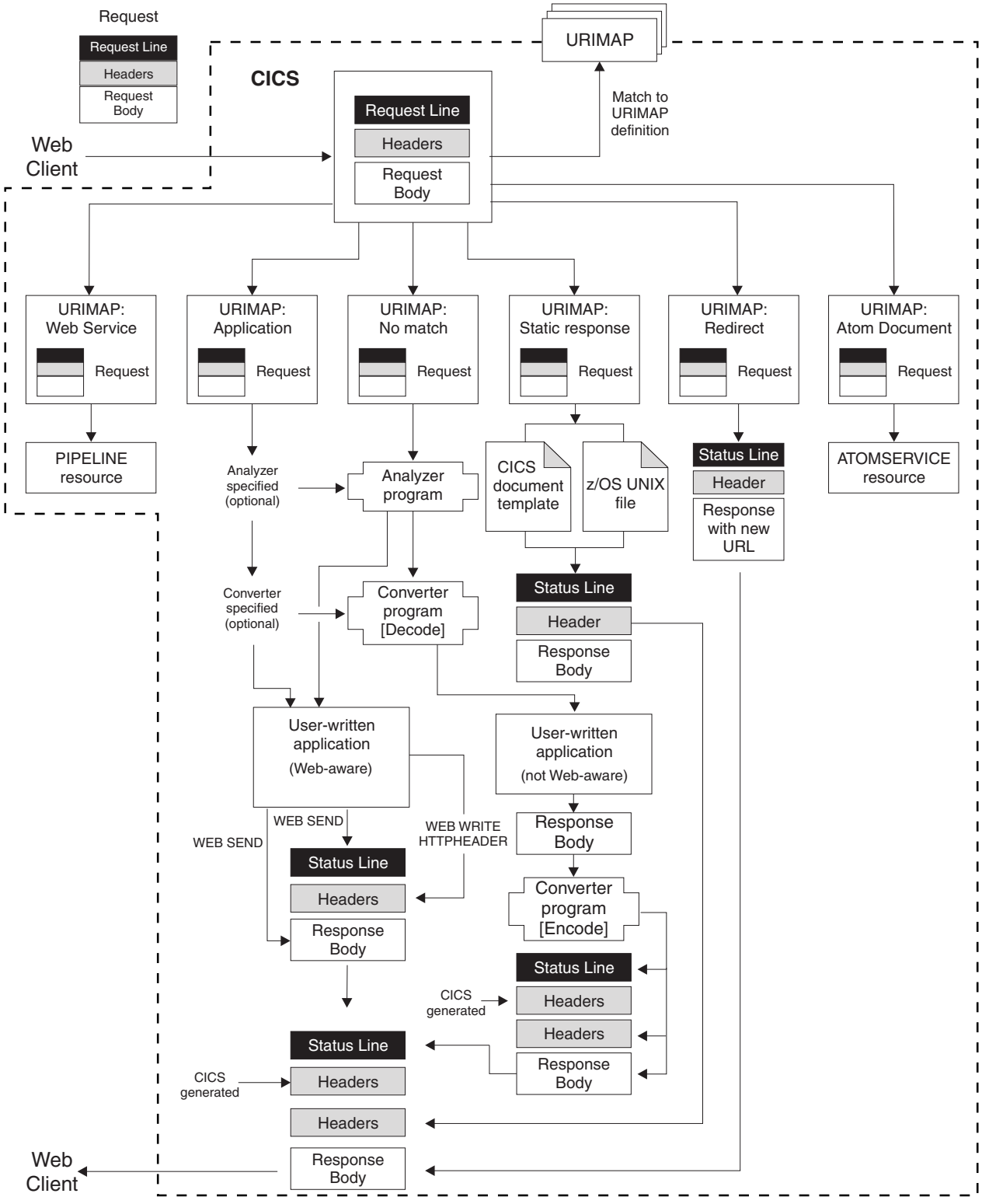


Figure 2. Processing for CICS as an HTTP server

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. The TCPIPSERVICE

definition specifies security attributes to be applied to the request, specifies the timeout setting for receiving the request message, and limits the maximum amount of data that can be received for a single request.

2. **CICS matches the URL for the request to a URIMAP definition, if available.** CICS tries to match the URL specified in the HTTP request to any URIMAP resource definitions that are related to the TCPIPSERVICE definition and apply to CICS as an HTTP server. If a successful match is made, the URIMAP definition tells CICS how to process the request. If no match is found, CICS continues with the default process, which begins at processing stage 5 with the analyzer program.
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. This completes the processing for that HTTP request.
4. **If the URIMAP definition relates to an Atom document, CICS locates the specified ATOMSERVICE resource to handle the request.** Processing for Atom documents is described in Chapter 20, "How Atom feeds work in CICS," on page 243.
5. **If the URIMAP definition relates to a Web service, CICS locates the specified PIPELINE resource to handle the request.** Processing for Web services is described in *CICS Web Services Guide*.
6. **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 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. This completes the processing for that HTTP request.
7. **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.

   The analyzer program for the TCPIPSERVICE definition must be used in the request processing path if no URIMAP definition has been set up for the request. It might also be needed if you are using a non-Web-aware application program that has special requirements, for code page conversion or for pre-CICS TS Version 3 compatibility processing. (Chapter 10, "Analyzer programs," on page 125 explains these situations.) Otherwise, the use of an analyzer program is optional, but note that the analyzer program is called to process the request if the URIMAP definition is not found.

   If an analyzer program is being used, the HTTP request and the HTTP headers are passed to the analyzer program. The analyzer program can interpret the request to determine:

   - Which CICS resources are to be used to service the request.
   - Which user ID is to be associated with the request.
   - Which of the remaining processing stages are required.
8. **A converter program may be used to decode the request and construct input to the application program.** Web-aware application programs should accept an HTTP request without any decoding. However, if you want to service an HTTP request using a non-Web aware application program that requires COMMAREA input, you can use a converter program to decode the request and construct input that fits the requirements of your application program. A converter program can be specified using a URIMAP definition, or it can be selected by an analyzer program.

9. **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.

   The application program runs under an alias transaction.

   The application program can perform the following tasks:

   - If the application program is Web-aware, it can examine the HTTP headers on the request, extract information (such as a query string) from the request line, receive the body of the request into a buffer for processing, select a status code and text for the status line of the response, and write HTTP headers for the response. **EXEC CICS** WEB API commands such as WEB SEND and WEB WRITE HTTPHEADER are used to construct the response.
   - Whether or not the application is Web-aware, it can produce output that forms the body of the response. Web-aware application programs can produce an entity body formed from a CICS document template or from a buffer of data. Application programs that are not Web-aware can produce output that can be converted by a Web-aware application program or a converter program into an entity body.

10. **A converter program may be used to encode the output from the application program and construct an HTTP response.** If the application program is not Web-aware and its output is not in the correct form to send to a Web client, you can use a converter program to produce an appropriate HTTP response including a status line and HTTP headers. The converter program can also perform other types of processing on the output, if desired.

    The converter program can specify that processing stages 6 (decoding or other processing using converter program), 7 (application program) and 8 (encoding or other processing using converter program) should be repeated. Because the converter program can change the name of the application program, you can use this facility to allow more than one application program to work on the same request in sequence, and provide a single response.

11. **If a request error or an abend occurs in the CICS Web support process, an error response is sent to the Web client, which can be customized using the user-replaceable Web error programs.** DFHWBEP or DFHWBERX receives information about the error situation, and the default HTTP response (including status code and status text) that CICS plans to send to the Web client. The user-replaceable programs can customize the response or build a new one, and return it to CICS for sending.

    The Web error programs are not used in all error situations. They are used when problems occur in initial processing of requests, and for abends or failures in subsequent processing. They are not used for situations where processing (such as processing by a user-written application program) completes correctly and an error or redirection response is the designed outcome.

12. **CICS generates some required HTTP headers and adds them to the message.** Appropriate headers are generated depending on the HTTP version for the response. If the response is HTTP/1.1, CICS adds headers that are required for HTTP/1.1 messages. If the response is HTTP/1.0, CICS adds the Connection: Keep-Alive header if the client has requested a persistent connection, and a small number of other headers. The values for some of

these headers are generated directly by CICS (such as the Date header), and the values of others are based on information provided by a Web-aware application program (using the WEB SEND command) or by a URIMAP definition. The headers can be added both to output from a Web-aware application, and to output from a converter program.

13. **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). "Code page conversion for CICS Web support" on page 39 explains when and how this takes place. The type of code page conversion that is required can be specified using options on the WEB SEND or WEB RECEIVE commands.

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. A Web-aware application program could be used to make an HTTP request, and then process the results to provide information to an application that is not Web-aware.

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.

Figure 3 on page 31 shows the process described in this topic.

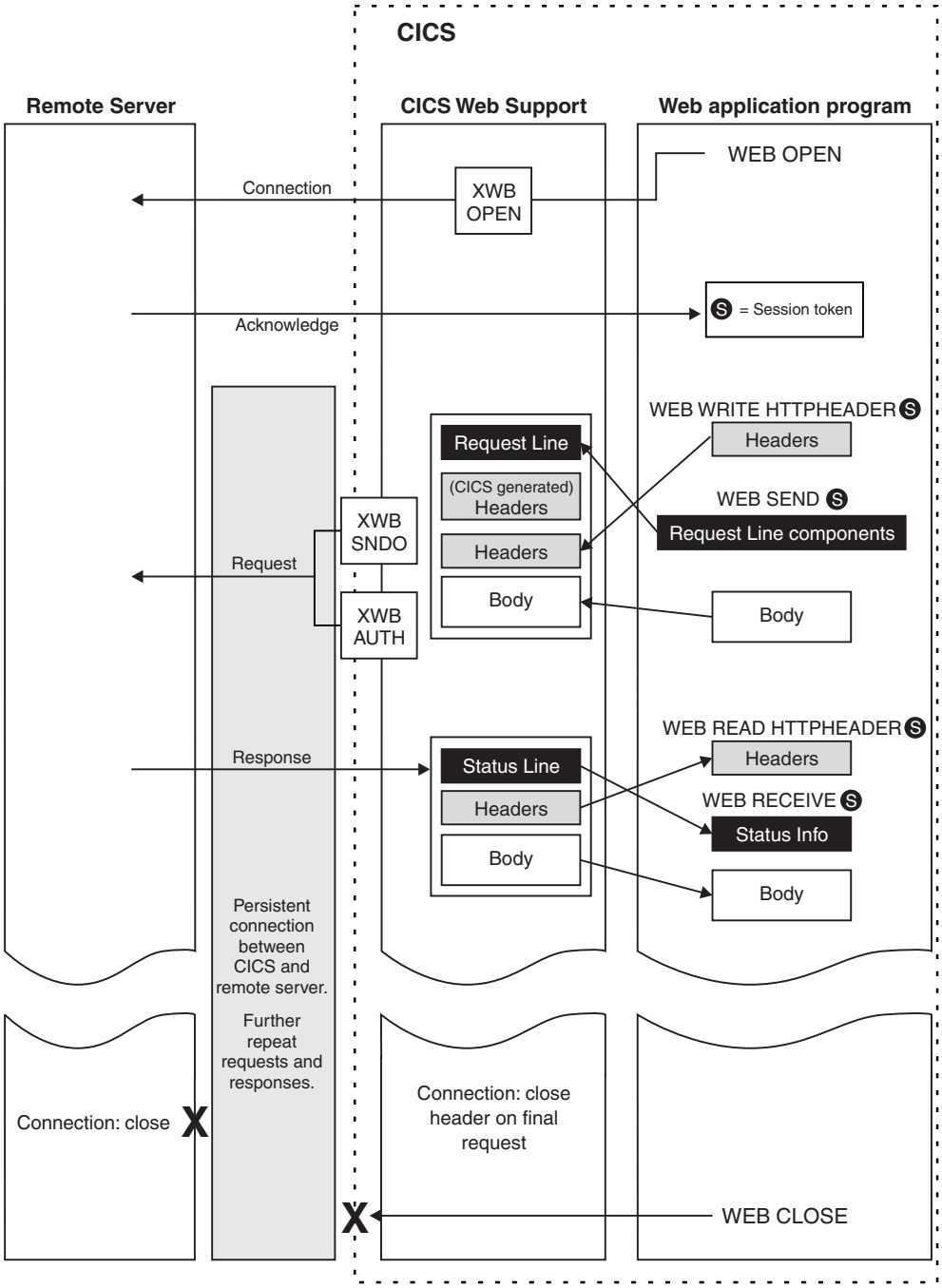


Figure 3. Processing for CICS as an HTTP client

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. (The *CICS Resource Definition Guide* has more information about URIMAP definitions.) An application may have more than one connection open at a time.
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. During this process, the XWBOPEN user exit may be used (if it has been activated using the ENABLE PROGRAM command) to redirect the application program's requests through a proxy server, if required, and to apply a security policy to connections to the host. 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. “Session tokens” on page 33 explains more about the session token.

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. The HTTP version for the request is supplied by CICS.
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. It can be formed from a CICS document (using the DOCUMENT interface), or from the contents of a buffer. If the server is at HTTP/1.1, chunked transfer-encoding may be used for a request body formed from the contents of a buffer (but not for a CICS document).
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.** The values for some of the headers are generated directly by CICS (such as the Date header), and the values of others are based on information provided by the application program (using the WEB SEND or WEB CONVERSE command) or by a URIMAP definition. During sending of the request, two user exits can be invoked, if required. XWBSNDO is called to apply a security policy for the individual request, and XWBAUTH specifies the username and password details required for Basic Authentication.
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). “Code page conversion for CICS Web support” on page 39 explains when and how this takes place. The type of code page conversion that is required can be specified using options on the WEB SEND, WEB RECEIVE or WEB CONVERSE commands. For CICS as an HTTP client, the default is that code page conversion does take place when messages are sent and received.

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 38 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.

URLs for CICS Web support

In a request URL for a resource that is provided by CICS Web support, the path component of the URL is up to you. In CICS Web support, the URIMAP definition or the analyzer program creates the linkage between the request URL and the resource provided by CICS, so the URL does not need to have any direct relationship to the CICS resource. However, you can design the URL to provide information for processing or administrative purposes.

“The components of a URL” on page 10 explains the different components of a request URL and their role.

A newer form of resource identifier, the IRI (Internationalized Resource Identifier), permits the use of characters and formats that are suitable for national languages other than English. An IRI can be used in place of a URI or URL where the applications involved with the request and response support this. CICS supports

the use of IRIs in URIMAP resource definitions. For more information about IRIs, see "Internationalized Resource Identifiers (IRIs)" on page 249.

URLs for application-generated responses

Information in a request URL can be used by analyzer programs and by user-written application programs.

Where an analyzer program is used in the processing path for the request, you can design a URL that tells the analyzer program which programs and transaction to specify for further processing. The CICS-supplied sample analyzer program DFHWBADX analyzes URLs with a path component in the format `/converter/alias/program/other path information`, where `converter` names the converter program (if any), `alias` names the alias transaction, `program` names the user application program, and `other path information` gives additional information that is not used by the analyzer.

A Web-aware application program which is providing a response can also use information from the path component of the URL. The path component can be extracted by the application using the WEB EXTRACT command, and analyzed to determine the appropriate action. For example, the path component can be used to specify a particular function provided by the application. Alternatively, if the Web-aware application is providing a front end for more than one other application, the path component of the URL can identify the application to which the request applies.

For application-generated responses that are managed using URIMAP definitions, the path components of URLs can be designed to map multiple request URLs to the same application. You can do this by making the path components of the URLs begin in the same way, and creating a single URIMAP definition with a wildcard to map all the request URLs to a single resource. For example, all requests whose path begins with `/staffapps/ordering/` could be mapped to a particular CICS application, by creating a URIMAP definition that specifies the path `/staffapps/ordering/*` and specifies the relevant application. The application can then extract and analyze information in the remainder of the URL to determine the appropriate action for each request.

URLs for static responses

In CICS Web support, the URL does not need to have any direct relationship to the CICS resource. For static responses, this means that the URL does not have to contain the full path to the file that provides the response. Instead, the URIMAP definition matches the request URL to the appropriate file.

However, where z/OS UNIX files are used as the static responses, you could decide to design the path components of the request URLs so that they match the directories used on z/OS UNIX. If all the z/OS UNIX files provided by CICS Web support are located in subdirectories of the same directory, such as the HOME directory of the CICS region userid, you might want to omit this directory and make the request URLs match the remainder of the paths to the files. For example, if your HOME directory is `/u/cts/CICSHome`, and you want to provide the following z/OS UNIX files as static responses:

```
/u/cts/CICSHome/FAQs/ordering.html
/u/cts/CICSHome/help/directory/viewing.html
```

you could use request URLs such as:

```
http://www.example.com/faqs/ordering.html
http://www.example.com/help/directory/viewing.html
```

Remember that the path components of URLs are case-sensitive, and so are z/OS UNIX names. URLs are normally specified in lower case. Take care to use the correct case when specifying each item in the URIMAP definition, especially if the file name is in mixed case and the URL is in lower case.

You might want to make your request URLs match your file directory structure:

- To make administration of resources more straightforward.
- To follow standard practice for Web servers.
- To reduce the number of URIMAP definitions that you need to create.

You can create a single URIMAP definition with wildcards, to deliver multiple static responses using the path matching mechanism. This is possible where the path component of the URL for all those static responses begins in the same way, and where the files for the static responses are stored in the same z/OS UNIX file directory. Wildcards are used at the end of the path component of the URL, and also at the end of the file path for the z/OS UNIX file. In the example above, all the HTML documents stored in the `FAQs` directory could be provided by a single URIMAP definition that specifies the path `/faqs/*` and specifies the HFSFILE attribute as `/u/cts/CICSHome/FAQs/*`. A similar technique can be used with CICS document templates whose names begin in the same way. Note that a URIMAP definition for a static response specifies a media type (for example `text/html`), so if you need to provide different file types in this way, ensure that they are stored in separate directories.

Query strings

A query string in a request URL can be used to select alternative URIMAP definitions. To use a query string for URIMAP matching, the complete and exact query string must be specified in the path attribute of the URIMAP definition, together with the path itself.

For application-generated responses, the application can extract and analyze information from a query string, using the WEB EXTRACT command or the WEB READ FORMFIELD command. This can be done whether or not the query string has been used for URIMAP matching.

If you are providing a static response with a document template, 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 can include appropriate variables in your document template to be substituted for the content of the query string. This happens only if the query string has not already been used for URIMAP matching.

URL length: CICS Web support

CICS Web support has the following limitations on URL length:

- For the URLs of inbound HTTP requests (for CICS as an HTTP server), CICS accepts a length of up to 32K. This length is at least eight times more than that supported by some commonly-used Web browser clients. If CICS does receive a URL that is longer than it can handle, it returns a 414 (Request-URI Too Long) status code.

- For the URLs of outbound HTTP requests (made by CICS as an HTTP client), CICS supports a path component of up to 255 characters in a URIMAP resource definition. The user application program that makes the request may override the URIMAP definition (or not use one at all), and supply a longer path component. Check the URL length that can be handled by the server.

URL length: URIMAP definitions

When choosing URLs for resources provided using URIMAP definitions, note the following additional limitations on URL length:

- CICS supports a path component of up to 255 characters in a URIMAP resource definition. Try not to use longer path components than this. The HTTP/1.1 specification says that servers should be cautious about URLs with a total length (comprising scheme, host and path components, and delimiters) that is greater than 255 characters, because older Web clients and proxies might not support these properly. If you are using an IRI that contains percent-encoded Unicode characters, note that a character in this context means a single ASCII character, not the original Unicode character. For example, the Cyrillic character that has the percent-encoded representation `%D0%B4` counts as 6 characters from the 255–character limit.
- If you need to use a longer path component, you usually can, because you do not have to specify the complete path in the URIMAP resource definition. An asterisk (*) may be used as a wildcard character at the end of the path. The behavior of the URIMAP definition will be correct if:
  - The specified part of the URL is unique to that resource.
  - The specified part of the URL is not unique to that resource, but you are providing a static response, and using the path matching mechanism to complete the URL.
- If you are using a query string for the purpose of URIMAP matching, and specifying it in the path attribute of the URIMAP definition, the total length must still be within the 255-character limit. (Part of the path component may be replaced by an asterisk, if the behavior will still be correct, but an asterisk cannot be used within the query string.) If you are not using the query string for this purpose, then any length of query string can be accepted, up to CICS Web support's overall 32K limit on URL length.
- For a redirection (using the LOCATION and REDIRECTTYPE attributes in the URIMAP definition), CICS supports a redirection URL of up to 255 characters. This must be a **complete** URL, including the scheme, host and path components, and appropriate delimiters. If you plan to use a resource as a destination for redirected clients, make sure that its complete URL fits within this 255-character limit.

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.

"Using chunked transfer-coding to send an HTTP request or response" on page 89 explains the process to use for chunked transfer-coding when sending an HTTP message from CICS. This procedure should be followed in order for your chunked message to be acceptable to the recipient.

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. This is to ensure that the responses are returned in the same order that the requests were sent. CICS treats each message in the pipelined sequence as a separate transaction, either providing a static response specified in a URIMAP definition, or passing the message to an application program and waiting for the application program to produce a response. 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. A server that supports pipelining provides the responses in the same sequence in which the requests were received. The application program begins to receive the responses when it has finished sending all its HTTP requests.

For CICS as an HTTP client, it is the application program's responsibility to ensure that any pipelined sequence of requests is idempotent. “Pipelining” on page 19 explains idempotency. For the benefit of your application program's logic as well as for the benefit of the server, if you are not sure that a sequence of requests is idempotent, it is advisable to make separate requests, and wait for a response to each request before sending the next one.

Related tasks:

“Sample programs: pipelining requests to an HTTP server” on page 158
Sample programs DFH$WBPA (Assembler), DFH$WBPC (C), and DFH0WBPO (COBOL) demonstrate how CICS can pipeline client requests to an HTTP server.

How CICS Web support handles persistent connections

Persistent connections are the default behavior 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.

If the application program needs to test whether the server has requested termination of the connection, the READ HTTPHEADER command can be used to look for the Connection: close header in the last message from the server. If the server requests closure of the connection, but the application program has not yet issued a WEB CLOSE command, CICS closes the connection but maintains the data relating to the connection (including the last response received from the server and its HTTP headers). The application program can continue to use that data until it issues a WEB CLOSE command or end of task is reached.

The WEB CLOSE command for CICS as an HTTP client does not cause CICS to notify the server that the persistent connection should be terminated. It only makes CICS close the connection. It is good behavior to include a Connection: close header on the final request that you make to the server. Using this header means that the server can close its persistent connection immediately after sending the final response, rather than waiting to see if CICS sends further requests and then timing out. To include this header, specify the CLOSESTATUS(CLOSE) option on the WEB SEND or WEB CONVERSE command.

If CICS as an HTTP client is communicating with an HTTP/1.0 server, CICS automatically sends Connection: Keep-Alive headers on HTTP messages. The application program that requested the connection does not need to provide these. Keep-Alive informs the server that a persistent connection is desired.

Code page conversion for CICS Web support

When CICS exchanges messages with a Web client or server, character data in the messages normally needs to undergo code page conversion on entering and leaving the CICS environment.

Code page conversion for text in messages is required for two reasons:

- CICS and user-written applications for CICS typically use an EBCDIC encoding, but Web clients and servers typically use an ASCII encoding.
- Within each encoding, a number of different code pages are used to support national languages.

Non-text content of messages, such as images or application data, does not require code page conversion.

In releases of CICS before CICS Transaction Server for z/OS, Version 3 Release 1, code page conversion for CICS Web support was handled using a code page conversion table (DFHCNV). In CICS Transaction Server for z/OS, Version 4 Release 1, the code page conversion table is no longer required for CICS Web support, except in limited circumstances for upgrade purposes. CICS Web support handles code page conversion using z/OS conversion services.

In CICS Web support, the defaults for code page conversion of text are:

- The default character set is the ASCII Latin-1 character set, ISO-8859-1. In HTTP messages, request or status lines and HTTP headers are typically in the US-ASCII character set, which is an older subset of ISO-8859-1. Message bodies containing text are often in ISO-8859-1.
- The default EBCDIC code page, for data in the CICS environment, is specified by the **LOCALCCSID** system initialization parameter for the CICS region. The default for **LOCALCCSID** is the EBCDIC Latin character set, code page 037.

Sometimes a more suitable alternative code page can be identified:

- A Web client or a server may specify a character set in the Content-Type header for a request or response, which is the character set that has been used for the message body.
- A Web client may send an Accept-Charset header on a request, stating which character sets are acceptable for the response.
- For non-HTTP requests and some older HTTP implementations, the character set used when transmitting the message might not be identified in the message headers, and you might need to identify this from your own knowledge of the message's source.
- Application programmers need to identify a suitable code page in which their application can receive message data, if the default is not suitable.

CICS does not support all the character sets named by IANA. The IANA character sets supported by CICS for code page conversion are listed in Appendix A, "HTML coded character sets," on page 373.

In most circumstances, the media type for the message can determine whether or not code page conversion takes place. Request or response bodies with a non-text media type usually do not undergo code page conversion. An exception is made for compatibility with Web-aware applications coded in earlier releases; if the options used on a command indicate that the application was coded before CICS Transaction Server for z/OS, Version 3 Release 1, the media type does not influence code page conversion.

Depending on the type of message and the processing path, code page conversion information might be identified automatically by CICS, or specified in the URIMAP definition, or specified by an analyzer program, or specified in the commands issued by a Web-aware application program. "Code page conversion for CICS as an HTTP server" explains the process for CICS as an HTTP server, and "Code page conversion for CICS as an HTTP client" on page 42 explains the process for CICS as an HTTP client.

Code page conversion for CICS as an HTTP server

When CICS as an HTTP server exchanges messages with a Web client, code page conversion is normally required for the message bodies. The method of specifying

this depends on whether you are making an application-generated response or a static response, and whether you are using a Web-aware application or a non-Web-aware application.

Request line and HTTP headers

Code page conversion for a request line or status line and for HTTP headers is handled as follows:

- Soon after receiving a request, CICS converts the request line (including any query string) and HTTP headers, from their character set, into the EBCDIC code page specified by the LOCALCCSID system initialization parameter (which applies to the whole of the local CICS region, and has a default of 037). For a successful conversion, you should set the LOCALCCSID system initialization parameter to any EBCDIC code page into which the ASCII Latin-1 character set ISO-8859-1 (code page 819) can be converted. If LOCALCCSID is set to an unsuitable code page, CICS uses the default EBCDIC code page 037 instead.
- When an application uses the WEB EXTRACT, WEB READ HTTPHEADER or WEB READ FORMFIELD commands to extract information from the request line (including any query string) and HTTP headers, the information is presented in its converted form, in the EBCDIC code page specified by the LOCALCCSID system initialization parameter (or the default 037).
- When CICS is preparing to send out a response, the status line and HTTP headers may be generated by CICS, or specified by the application using the WEB WRITE HTTPHEADER command. Before sending, all the headers and the status line are converted from the EBCDIC code page in which they were specified, into the US-ASCII character set.

Message bodies: application-generated response

If the request is to have a dynamic response from a user-written application, code page conversion for message bodies is handled as follows:

- If a Web-aware application receives the request, CICS carries out code page conversion if any of the code page conversion options are used to specify conversion on the WEB RECEIVE command. If none of the options are present, code page conversion does not take place. You can either supply, or allow CICS to identify, the character set, and request a code page if the default is not suitable.
- If an analyzer program is used in the processing path for the request, the analyzer program can specify or suppress code page conversion for the copy of the request which is passed to subsequent processing stages in a block of storage. You supply both the character set and the application code page that are used. CICS still holds the original version of the request body. Applications or converter programs which use the **EXEC CICS** WEB API commands access the original body, not the block of storage, and they can specify code page conversion on the **EXEC CICS** WEB API commands.
- When a converter program is passed the request in a block of storage, if there is no analyzer program in the processing path, CICS converts the request body in the block of storage, identifying the character set and converting to the default code page.
- To identify the character set that the Web client used for the request body, CICS examines the request headers. If the request headers do not provide this information, or the specified character set is unsupported, CICS assumes as a default that the message body is in the ISO-8859-1 character set. If the message

body is not in that character set, and there is no information in the headers, you need to identify the correct character set.

- By default, CICS converts the request body into the EBCDIC code page specified by the LOCALCCSID system initialization parameter (which applies to the whole of the local CICS region, and has a default of 037). If your application requires a different code page (which could be EBCDIC or ASCII), you can specify this.
- If an application or converter program sends the response using the **EXEC CICS** WEB API commands, CICS carries out code page conversion if any of the code page conversion options are used to specify conversion on the WEB SEND command. If none of the options are present, code page conversion does not take place.
- If a converter program produces the response in a block of storage and passes it to CICS for sending, CICS mirrors the code page conversion that was carried out for the request. The character set and host code page settings from the analyzer program, or the default settings in the absence of an analyzer program, are used. If the analyzer program suppressed code page conversion for the request, no code page conversion is carried out for the response body.

Message bodies: static response

If the request is to have a static response determined by a URIMAP definition, code page conversion for message bodies is handled as follows:

- For a static response, CICS does not examine any message body that is present on a Web client's request, so no code page conversion is required.
- You specify code page conversion for the body of the response in the URIMAP definition that produces the static response. If the response contains text, the URIMAP definition needs to specify all of the following:
  - A text media type, using the MEDIATYPE attribute. There is no default for this attribute.
  - A character set for the Web client, using the CHARACTERSET attribute.
  - The code page in which the CICS document template or z/OS UNIX file for the response is encoded, using the HOSTCODEPAGE attribute.

CICS retrieves the z/OS UNIX file, or retrieves the CICS document template and creates the document, and then carries out appropriate code page conversion.

Code page conversion for CICS as an HTTP client

When CICS as an HTTP client exchanges messages with a server, code page conversion is normally required for the message bodies. You specify an application code page when opening the connection. The character sets can usually be identified by CICS or allowed to default.

Request line and HTTP headers

Code page conversion for a request line or status line and for HTTP headers is handled as follows:

- When CICS is preparing to send out a request, the request line and HTTP headers may be generated by CICS, or specified by the application using the WEB WRITE HTTPHEADER command. Before sending, all the headers are converted from the EBCDIC code page in which they were specified, into the US-ASCII character set.

- Soon after receiving a response, CICS converts the status line and HTTP headers from the US-ASCII character set, into the EBCDIC code page 037. The application receives the status line and other information, and examines the HTTP headers, in their converted form, in the EBCDIC code page 037.

Message bodies

Code page conversion for the message bodies is handled as follows:

- The EBCDIC code page used by the application program is specified on the WEB OPEN command that initiates communication with the server. The default is the EBCDIC code page specified by the LOCALCCSID system initialization parameter (which applies to the whole of the local CICS region, and has a default of 037). CICS uses this information for converting the message bodies for requests and responses on this connection.
- For each request that the application sends out, the CLIENTCONV option on the WEB SEND or WEB CONVERSE command specifies whether or not CICS carries out code page conversion for the request body. The default is that code page conversion does take place. If you are using the WEB CONVERSE command, you can choose to specify code page conversion for either, both, or neither of the request body and the response body.
- If you have specified conversion for a request, the default is that CICS converts the request body to the ISO-8859-1 character set. You can use the CHARACTERSET option on the WEB SEND or WEB CONVERSE command to select an alternative, if you know that the server prefers a different character set.
- For each response that the application receives, the CLIENTCONV option on the WEB RECEIVE or WEB CONVERSE command specifies whether or not CICS carries out code page conversion for the response body, into the EBCDIC code page specified when the connection was opened. The default is that code page conversion does take place. CICS examines the response headers to identify the character set that the server used for the response body. If the response headers do not provide this information, or the named character set is unsupported, CICS assumes as a default that the message body is in the ISO-8859-1 character set.

HTTP/1.1 compliance for CICS as an HTTP server

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* (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 provided by CICS itself. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be unconditionally compliant. An implementation is not

compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements.

There are three aspects to CICS Web support compliance with the HTTP/1.1 specification.

- CICS Web support performs actions that are required from an HTTP server. For example, CICS Web support receives inbound requests, maintains persistent connections, writes certain HTTP headers, and transmits responses. The behavior of CICS Web support during these actions is conditionally compliant with the HTTP/1.1 specification. Where necessary, this represents a change in behavior from earlier releases of CICS. "CICS Web support behavior in compliance with HTTP/1.1" describes how the behavior of CICS Web support complies with HTTP/1.1, and where this differs from releases of CICS before CICS Transaction Server for z/OS, Version 3 Release 1.
- Some parts of the HTTP/1.1 specification are not relevant to CICS Web support. For example, CICS Web support does not act as a proxy server or provide a caching facility. "HTTP functions not supported by CICS Web support" on page 46 notes these areas, so that you know the areas in which HTTP/1.1 compliance is not a concern for CICS Web support and for your user application program.
- Web-aware user application programs in CICS can be used to create application-generated HTTP responses and instruct CICS Web support how to serve the responses. If you want your CICS Web support implementation to be compliant with the HTTP protocol specifications, in particular HTTP/1.1, your user application programs share the responsibility for compliance in the actions that they perform. Some basic guidance information is provided in this documentation, but it is important to check the HTTP specification to which you are working for more detailed information. Chapter 6, "Writing Web-aware application programs for CICS as an HTTP server," on page 75 describes the process for writing a user application program for CICS Web support.

  In practical terms, the different levels of requirement in the HTTP/1.1 specification (MUST, SHOULD, or MAY) should be handled by your application program as follows:

  – MUST level requirements must be implemented to maintain compliance. CICS Web support is designed to handle, or to assist you to comply with, all the MUST level requirements that apply to straightforward activities. Some additional MUST level requirements might apply if you choose to fulfil an optional requirement that CICS Web support does not already handle. Also, some MUST level requirements relate to actions that your application must *not* perform in certain circumstances.
  – SHOULD level requirements are not necessary for conditional compliance with the HTTP specifications. CICS does not comply with all the SHOULD level requirements in the HTTP/1.1 specification. However, if your application can comply with a SHOULD level requirement without inconvenience, it is advisable to do so.
  – MAY actions are optional for any HTTP implementation, whatever its level of compliance. They should be treated as suggestions or recommendations.

CICS Web support behavior in compliance with HTTP/1.1

CICS Web support complies on your behalf with many of the requirements in the HTTP/1.1 specification.

Most of the behavior described here applies whether you are using URIMAP definitions or an analyzer program to handle HTTP requests for CICS as an HTTP server, but there are a few exceptions.

- **CICS checks inbound messages for compliance with HTTP/1.1, and handles or rejects non-compliant messages.** The checks are made immediately on receipt of the request, before a URIMAP definition or analyzer program is involved. A variety of basic acceptance checks are made, and if the message is unacceptable and it is not appropriate for CICS to handle the problem itself, an error response is returned to the Web client where possible. These basic acceptance checks are not carried out for HTTP/1.0 messages, nor are they carried out if the USER protocol (instead of the HTTP protocol) is specified on the TCPIPSERVICE definition.

  Note: CICS requires the Content-Length header on all inbound HTTP/1.1 messages that have a message body. If a message body is present but the header is not provided, or its value is inaccurate, the socket receive for the faulty message or for a subsequent message can produce unpredictable results. For HTTP/1.0 messages that have a message body, the Content-Length header is optional.
- **CICS follows the HTTP/1.1 rules for comparison of URLs.** Scheme names and host names are compared case-insensitively, but paths are compared case-sensitively. All components are unescaped before comparison. CICS also handles the absolute URI form in requests (where the host name is specified in the request line). Note that when you use an analyzer program instead of a URIMAP definition to handle an inbound HTTP request, if you need to achieve compliance in this respect, you must code your analyzer program to perform URL comparison according to the rules stated in the HTTP/1.1 specification. (See "The HTTP protocol" on page 12 for more information about the HTTP specifications.)
- **CICS provides a suitable HTTP version number in the start line of outbound messages.** CICS normally identifies the message as HTTP/1.1, unless it knows that the Web client or server is at HTTP/1.0 level. In that case, CICS identifies the message as HTTP/1.0. Requests by a Web client to upgrade from one HTTP version to another, or to a different security protocol, are not supported by CICS.
- **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.** Some headers are also produced by CICS which are not required for compliance, but support actions that the application program has requested (such as the Expect header). Appendix B, "HTTP header reference for CICS Web support," on page 375 describes the headers that CICS writes and the circumstances in which these headers are created. The headers for compliance are supplied for messages sent by both Web-aware applications and non-Web-aware applications. They are not supplied if the USER protocol (instead of the HTTP protocol) is specified for the TCPIPSERVICE definition. For HTTP/1.0 messages, only the Connection: Keep-Alive, Content-Length, Date and Server headers are supplied.
- **CICS takes action on the Expect header for both inbound and outbound requests.** When CICS as an HTTP server receives a request with the Expect header, it sends a 100-Continue response to Web client and waits for the remainder of the request. For CICS as an HTTP client, you can use the EXPECT option on the WEB SEND command to make CICS send the Expect header to the server and wait for the 100-Continue response before sending the request. If the server returns a different response, CICS informs the application program and cancels the send.
- **CICS handles OPTIONS requests from Web clients and makes an appropriate response.** OPTIONS * (an inquiry on the whole server, rather than a specific resource) is the only format accepted. The response contains basic information about CICS as an HTTP server (the HTTP version and server software description). No user application program is involved.

- **CICS handles TRACE requests from Web clients and makes an appropriate response.** When CICS Web support receives a correctly formed request with the TRACE method, it returns a response containing the request with its original headers, plus any headers it acquired (such as the Via header). No user application program is involved.
- **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.** Trailing headers for chunked messages can be manipulated through the HTTP header read, write and browse commands. This means that applications can receive and handle chunked messages as they would normal messages. CICS also supports sending chunked messages out from a user application, but you must ensure that you follow the correct procedure to make the chunked message acceptable to the recipient. "How CICS Web support handles chunked transfer-coding" on page 36 explains CICS Web support behavior in this respect.
- **CICS supports pipelining for both inbound and outbound messages.** CICS lets you receive pipelined requests from a Web client. CICS passes each request to the application program in turn, to ensure that the application complies with HTTP/1.1 by responding to the requests in the order they were received. CICS also lets you send pipelined requests to a server, but you must ensure that you follow the correct procedure to make the pipelined request sequence compliant with HTTP/1.1. "How CICS Web support handles pipelining" on page 37 explains CICS Web support behavior in this respect.
- **CICS supports virtual hosting (multiple host names at the same IP address).** Support for virtual hosts is based on your URIMAP definitions. "Virtual hosting" on page 9 explains the support provided.
- **Connections are persistent by default.** This is the case for both CICS as an HTTP server and CICS as an HTTP client. If CICS is communicating with a Web client or server at HTTP/1.1 level, it keeps connections open unless the Web client, the server, or the user application in CICS, specifically requests closure, or the task ends. If CICS is communicating with a Web client or server at HTTP/1.0 level, it sends Connection: Keep-Alive headers when a persistent connection is supported. "How CICS Web support handles persistent connections" on page 38 explains CICS Web support behavior in this respect.

HTTP functions not supported by CICS Web support

The HTTP/1.1 specification defines various roles for the parties that make use of the HTTP protocol. CICS Web support carries out many of the functions 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 proxies, gateways, tunnels and caches, and these requirements are not relevant to CICS Web support and can be ignored.

- **CICS does not act as a proxy.** You can ignore all the requirements in the HTTP/1.1 specification that relate to the behavior of proxies.
- **CICS does not act as a gateway (an intermediary for another server) or a tunnel (a relay between HTTP connections).** You can ignore all the requirements in the HTTP/1.1 specification that relate to the behavior of gateways and tunnels.
- **CICS does not provide caching facilities, or provide support for user-written caching facilities.** You can ignore all the requirements in the HTTP/1.1 specification that relate to the behavior of caches. Although you may store any information you receive from a server, you should be careful that you do not

deliver the stored information to a user who is making a request in the expectation of receiving current information from the server.

- **CICS is not designed for use as a Web browser.** Through CICS as an HTTP client, user application programs can make requests for individual, known resources that are available from a server, but they would not be expected to browse the Internet generally. CICS does not provide history lists, favorites lists, or other features of a Web browser, so any requirements relating to these can be ignored.

See "The HTTP protocol" on page 12 for more information about the HTTP specifications.

Atom format and publishing protocol compliance

CICS supports the Atom format and Atom publishing protocol described in RFC 4287 and RFC 5023.

The Atom Syndication Format and Atom Publishing Protocol are described in the following Internet Society and IETF (Internet Engineering Task Force) Request for Comments documents (RFCs):

RFC 4287, *The Atom Syndication Format*
: The specification for the structure of Atom feed and entry documents. You can read this specification at http://www.ietf.org/rfc/rfc4287.txt.

RFC 5023, *The Atom Publishing Protocol*
: The protocol for publishing and editing resources in the Atom document format over HTTP, including descriptions of the structure of Atom service and category documents. You can read this specification at http://www.ietf.org/rfc/rfc5023.txt.

CICS behavior in compliance with the Atom format and protocol

CICS handles a number of tasks required for compliance with the Atom format and protocol.

- CICS validates the format of your Atom configuration files for Atom feed documents, Atom entry documents, and collections to ensure that they comply with the requirements in RFC 4287. Wherever it is viable to do so, CICS checks that you have supplied required elements, and applies defaults where necessary. The CICS documentation for setting up Atom feeds has information about the situations in which CICS is not able to validate your data, where you are responsible for providing correct data.
- CICS handles Atom service documents and Atom category documents that you create, and delivers these to clients. However, CICS does *not* validate the format of these documents, whether they are in Atom configuration files or delivered as static documents. You are responsible for compliance with the requirements detailed in RFC 5023 for the format of these Atom document types.
- CICS provides EXEC CICS API commands that create timestamps in the correct format for Date constructs in Atom documents. If you choose not to use the EXEC CICS API commands to produce your timestamps, you are responsible for compliance with the requirements detailed in RFC 4287 for the content of Date constructs.
- CICS handles the mapping and comparison tasks for IRIs, which are used in <atom:id> elements and <atom:link> elements.

- CICS supplies the <atom:generator> element for Atom feed documents, which gives the name of the generating agent, with a URI and version number.
- CICS can generate an Atom ID for each entry when it serves the Atom feed, in a format that meets the specification in RFC 4287. Under the conditions described in “Atom IDs for Atom entries” on page 255, the Atom ID is unique and remains the same for the lifetime of each Atom entry.
- CICS carries out the actions required of a server for processing client requests to manipulate ordinary Atom entries in collections, as specified in RFC 5023, except where noted in “Atom features not supported by CICS.” In particular, CICS does not support media resources and media link entries in collections.
- CICS enables you to implement security for Atom feeds and resources containing data for Atom feeds, through CICS resource security and CICS Web support security. You are responsible for choosing and setting up appropriate security features to protect your Atom feeds. RFC 5023 recommends that you use authentication mechanisms, and discusses a number of threat scenarios.

Atom features not supported by CICS

Some of the requirements for the Atom format and protocol are not relevant to CICS or not supported by CICS.

- The requirements in RFC 4287 and RFC 5023 for "Atom Processors" are not relevant to CICS. CICS does not receive Atom feed documents for processing, so it does not act as an Atom Processor.
- CICS does not provide support for digital signatures and encryption of Atom documents, but, in compliance with RFC 4287, CICS does not reject an Atom document that contains a signature.
- CICS does not support HTML or XHTML content in a Text construct (such as the <atom:title> element). You must supply plain text content with no child elements.
- CICS permits HTML, XHTML, and other text media types (such as XML) as content in the <atom:content> element. However, you must validate the content yourself. CICS does not attempt to parse the content to check that its markup is valid, but supplies it to the client as you provided it.
- It is possible to use the <atom:content> element to supply content with a non-text media type if the content is in a valid Base64 encoding. CICS does not disallow this, but provides no support for non-text content, and does not check for the presence of an <atom:summary> element, which is required for content encoded in Base64.
- CICS does not support media resources and media link entries in collections. Media resources are specified by the Atom Publishing Protocol (RFC 5023) primarily as a means of organizing nontext content in a collection. When a client attempts to create an entry in a collection, CICS rejects with a 415 status code any client request that is not an Atom entry (with the media type application/atom+xml, with or without the type=entry parameter). Do not specify any additional media types in <app:accept> elements in a service document for CICS.
- The support for Atom feeds in CICS is intended for supplying feeds of data to clients. CICS does not support the delivery of Atom entries that contain no data, such as entries that use the "src" attribute to reference remote content.
- CICS does not support the optional <atom:source> element for an entry. The <atom:source> element is used to preserve metadata if an Atom entry is copied from one feed into another.

- CICS only supports the link elements <atom:link rel="self"> and <atom:link rel="edit">. Other link elements, such as related, enclosure, or via links, and links to alternate formats, are not supported. CICS does not support title or length attributes for link elements.
- CICS does not reject Atom entries for a collection on the basis of categories. You can use the <app:categories> element in a service document or category document to specify acceptable categories for entries in a collection, but CICS does not police whether clients adhere to these categories.
- CICS ignores Slug headers and supplies its own URIs for Atom entries.
- For reasons of performance, CICS does not automatically return Atom entries in a collection in the order in which they were most recently edited (as shown by the <app:edited> element in the entry). This function is a SHOULD requirement in RFC 5023 for a full list of entries, but a MUST requirement for a partial list of entries. CICS deviates from this requirement in order to maintain acceptable response times while still providing the useful function of partial lists. If you are using a service routine to supply the entries to CICS, you can make the collection compliant by supplying the entries in the order in which they were last edited, if your resource is able to store this information.
- CICS does not support <app:control> and <app:draft> elements in Atom entries, and ignores them.
- Some resources that provide Atom entry data for CICS cannot follow the recommendation in RFC 4287 to store Atom IDs with Atom entries. However, CICS has the capability to generate the same Atom ID for an Atom entry each time the entry is served, under the conditions described in "Atom IDs for Atom entries" on page 255.
- RFC 4287 requires that the Atom ID remain with the Atom entry if the entry is reused or moved to another location. If you store Atom IDs with your Atom entries, you can move the Atom entries to another location and still comply with this requirement. If you do not store Atom IDs with your Atom entries, do not move the Atom entry to another location.

Part 2. CICS Web support

Information about planning, configuring and managing a CICS Web support architecture, and writing application programs and utility programs for CICS Web support.

Chapter 4. Configuring CICS Web support base components

The base components of CICS Web support are needed for all CICS Web support tasks. You need to configure these before starting to work with CICS Web support.

About this task

"Components of CICS Web support" on page 22 lists all the components. The base components that you need to set up are:

- TCP/IP support
- Access to z/OS UNIX System Services
- SSL support
- System initialization parameters

If you want to use an analyzer program that you coded in an earlier CICS release to reference the code page conversion table DFHCNV, you might need to set up some DFHCNV entries. Code page conversion table entries are not required for new CICS Web support development.

When you have set up these base components, you can verify the operation of CICS Web support using the supplied sample programs.

Procedure

1. Enable TCP/IP support for the CICS region, following the instructions in the *CICS Transaction Server for z/OS Installation Guide*. This process includes setting up Communications Server and establishing access to a DNS, or domain name, server through z/OS.
2. Enable CICS to access z/OS UNIX System Services by including an OMVS segment in the user profile of the CICS region user ID, following the instructions in Giving CICS regions access to z/OS UNIX System Services the *CICS Transaction Server for z/OS Installation Guide*.
3. Set up SSL support, following the instructions in the *CICS RACF Security Guide*. The *CICS RACF Security Guide* also explains the facilities that SSL provides.
4. Specify appropriate system initialization parameters, following the steps in "Specifying system initialization parameters for CICS Web support" on page 54. Some of the system initialization parameters are optional at this stage.
5. Reserve as many ports belonging to z/OS Communications Server as you need for CICS Web support. "Reserving ports for CICS Web support" on page 54 has more information about this.
6. Optional: If you have existing request processing structures in CICS Web support that include an analyzer program which references entries in the conversion table (DFHCNV), and you want to continue using these request processing structures unchanged, "Upgrading entries in the code page conversion table (DFHCNV)" on page 55 tells you what to do. You do not need a code page conversion table for any other CICS Web support tasks.
7. Verify the operation of CICS Web support using the supplied samples. "Verifying the operation of CICS Web support" on page 56 tells you how to do this.

Specifying system initialization parameters for CICS Web support

You need to specify certain system initialization parameters to enable CICS Web support.

About this task

Procedure

1. Required: Specify `TCPIP=YES` to activate CICS TCP/IP services. The default setting is NO. YES must be specified to enable CICS Web support.
2. Use the **LOCALCCSID** system initialization parameter to specify the coded character set identifier for the local CICS region. This is the code page that CICS considers as the default for application programs. The default is the EBCDIC code page 037. If no alternative code page conversion options are selected, CICS will translate the data content of incoming HTTP requests into this code page before passing it to an application program, and will assume that the application has provided HTTP responses in this code page.
3. If you are planning to use CICS document template support, either to provide a static response to HTTP requests, or as part of an application-generated response, specify the default host code page for the document domain using the **DOCCODEPAGE** system initialization parameter. The default is the EBCDIC code page 037.
4. If you are planning to give Web clients access to 3270 display applications, specify suitable timeout periods using the **WEBDELAY** system initialization parameter. The two timeout periods control:
   - The length of time, in minutes, after which a Web task and its associated data is marked for deletion if no activity takes place on it. The default is 5 minutes.
   - The frequency, in minutes, with which the garbage collection transaction CWBG is run to delete the marked tasks and their data. The default is 60 minutes.

   WEBDELAY does not apply to any other CICS Web support tasks that do not involve 3270 display applications.
5. If you are want to use security with CICS Web support, you need to also set values for the following additional system initialization parameters:
   - **CRLPROFILE**
   - **ENCRYPTION**
   - **KEYRING**
   - **MAXSSLTCBS**
   - **SSLCACHE**

   The *CICS RACF Security Guide* explains how to make SSL work with CICS, including specifying these system initialization parameters.

Reserving ports for CICS Web support

You are recommended to reserve as many ports belonging to z/OS Communications Server as you need for CICS Web support, and to ensure that CICS Web support has exclusive use of those ports where possible.

Procedure

- For HTTP, the well known (or default) port number is 80, and for HTTPS, the well known port number is 443. You should take care to resolve conflicts with any other servers at the same IP address that might use the well-known ports.
- Application programmers may use port numbers from 1024 to 32 767 for nonstandard servers. Ports below 1024 are the well known port numbers which are architected by IANA for particular functions, so except for the HTTP port 80 and the HTTPS port 443, these should not be used for CICS Web support. SSL and non-SSL requests must use separate ports.
- To reserve a port on which CICS Web support listens for incoming client requests, you can specify the PORT statement or the CICS job name in the PROFILE.TCPIP data set, as described in *z/OS Communications Server: IP Configuration Reference*, SC31-8776.
- The maximum length of any queue of requests for a TCP/IP port on which a program is listening is controlled by the SOMAXCONN parameter in the PROFILE.TCPIP data set. CICS listens on a TCP/IP port, so you must coordinate the value of this parameter with the value chosen for the BACKLOG parameter in the TCPIPSERVICE definition.

Upgrading entries in the code page conversion table (DFHCNV)

In CICS Transaction Server for z/OS, Version 4 Release 1, a code page conversion table is not normally required for CICS Web support. However, if you have existing request processing structures that include an analyzer program which references entries in the conversion table, and you do not want to change the analyzer program, you can continue to provide DFHCNV entries.

About this task

In releases of CICS before CICS Transaction Server for z/OS, Version 3 Release 1, the code page conversion table (DFHCNV) was used to define code page conversions between the code pages used within CICS, and the ASCII code pages used by Web clients. For CICS Web support in CICS Transaction Server for z/OS, Version 4 Release 1, you do not need to create any new entries in the code page conversion table. CICS Web support handles code page conversion using z/OS conversion services.

However, 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 upgrade your DFHCNV entries. “Writing an analyzer program” on page 128 tells you how to code your output parameters in this way.

Note: As supplied, the CICS-supplied sample analyzer DFHWBADX specifies an entry defined in the sample code page conversion table DFHCNVW$. The sample conversion table can be used without any configuration, but you might prefer to modify DFHWBADX to use the new output parameters, to provide greater control and avoid the use of the sample conversion table.

If you prefer to continue using DFHCNV, follow these steps:

Procedure

1. Locate your source for the DFHCNV resource definition macros that you used to define the conversion table in an earlier CICS release. The sequence of macros should include a DFHCNV TYPE=ENTRY macro for each pair of code pages.
2. Use the macros to set up a DFHCNV conversion table, following the process described in the *CICS Intercommunication Guide*. You need to define, assemble and link-edit the table.

Verifying the operation of CICS Web support

Sample programs DFH$WB1A (Assembler) and DFH$WB1C (C) help you to test that CICS Web support is working. The sample programs use EXEC CICS WEB and DOCUMENT commands to receive your request and construct and send a simple response.

About this task

You can access DFH$WB1A using the CICS-supplied sample analyzer program DFHWBADX. You can access DFH$WB1C using the supplied sample URIMAP definition DFH$URI1 or the sample analyzer program. If you plan to use CICS as an HTTP client, note that the CICS-supplied sample programs for pipelining client requests work with a CICS region that has DFH$WB1C and DFH$URI1 set up, so you might want to choose this option now.

The sample programs construct HTTP responses like this:

```
DFH$WB1A on system applid successfully invoked through CICS Web support
```

where *applid* is the applid of the CICS system in which CICS Web support is running.

To run the sample programs:

Procedure

1. Modify as necessary and then install the sample TCPIPSERVICE definition HTTPNSSL, which is provided in group DFH$SOT. The CICS-supplied sample analyzer program DFHWBADX is specified in the TCPIPSERVICE definition. You might need to change the following options:
   a. **PORTNUMBER**: HTTPNSSL uses port 80, the well-known port number for HTTP. If port 80 is not reserved for the use of CICS, specify another port belonging to z/OS Communications Server that you have reserved for the use of CICS.
   b. **HOST** or **IPADDRESS**: HTTPNSSL does not specify an IP address, so this option defaults to the IP address corresponding to the default z/OS Communications Server TCP/IP stack. This situation is the most usual. If you have multiple TCP/IP stacks in your z/OS image, and you want to use a nondefault stack, specify the IP address corresponding to that stack.
2. If you want to use the sample program DFH$WB1C, install its PROGRAM resource definition, which is provided in the DFH$WEB resource definition group. The PROGRAM resource definition for DFH$WB1A is in the DFHWEB resource definition group, which is already installed as part of DFHLIST.
3. If you are using the sample program DFH$WB1C, and you want to try using a URIMAP definition, install the supplied sample URIMAP definition DFH$URI1, which is provided in the DFH$WEB resource definition group.

4. At a Web browser, enter a URL that connects to CICS Web support using the following URL components:

   Scheme
   HTTP

   Host The host name assigned to the z/OS image. If you do not know the host name, you can use the dotted decimal IP address from the HTTPNSSL TCPIPSERVICE definition. If you did not specify the IP address explicitly, it has been filled in by CICS, and you can view it in the installed TCPIPSERVICE definition.

   Port number
   The port number specified in the TCPIPSERVICE definition. If the number is 80, you do not have to specify it explicitly.

   Path
   - To access DFH$WB1A, use the path `/CICS/CWBA/DFH$WB1A`
   - To access DFH$WB1C, use the path `/sample_web_app` if you have installed the sample URIMAP definition, or the path `/CICS/CWBA/DFH$WB1C` if you want to use the sample analyzer program instead.

5. Unless you want to carry out further testing now, uninstall the sample TCPIPSERVICE definition HTTPNSSL and disable the URIMAP definition DFH$URI1. You can replace HTTPNSSL with your own properly architected TCPIPSERVICE definition later on.

Related tasks:

"Sample programs: pipelining requests to an HTTP server" on page 158
Sample programs DFH$WBPA (Assembler), DFH$WBPC (C), and DFH0WBPO (COBOL) demonstrate how CICS can pipeline client requests to an HTTP server.

Configuring the HTTP TRACE method

By default, all HTTP TRACE requests receive an HTTP 200 (OK) response. You can override this setting so that trace is not enabled.

About this task

You will change the HTTP TRACE request setting so that an HTTP TRACE request receives an HTTP 501 (Not Implemented) response.

Procedure

Create an assembler data-only module, DFHWBMTH, which contains a halfword length followed by a 7 byte field that contains the characters 'NOTRACE'.

Example

Here is an example to help you create your module:

```
//DFHWBMTH  JOB   'accounting info',name,MSGCLASS=A
      //ASM      EXEC PGM=ASMA90,REGION=2048K,
      //            PARM=(DECK,NOOBJECT,ALIGN)
      //SYSPRINT DD SYSOUT=*
      //SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
      //SYSUT1   DD SPACE=(CYL,(3,2))
      //SYSUT2   DD SPACE=(CYL,(1,1))
      //SYSUT3   DD SPACE=(CYL,(1,1))
      //SYSPUNCH DD DSN=&&OBJMOD,DISP=(,PASS),
```

```
//            SPACE=(CYL,(1,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//SYSIN    DD DATA,DLM='@@'
DFHWBMTH CSECT
DFHWBMTH AMODE 31
DFHWBMTH RMODE ANY
LENGTH   DC AL2(ENDDATA-*)
OPTIONS  DC CL7'NOTRACE'
ENDDATA  EQU *
         END     DFHWBMTH
@@
//LKED     EXEC PGM=IEWL,REGION=2048K,
//            PARM=(LIST,XREF),
//            COND=(7,LT)
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD SPACE=(CYL,(1,1))
//SYSLIN   DD DSN=&&OBJMOD,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//SYSLMOD  DD DSN=CICS.DFHRPL,DISP=SHR
//SYSIN    DD DATA,DLM='@@'
    MODE AMODE(31),RMODE(ANY)
    NAME DFHWBMTH(R)
@@
```

Chapter 5. Planning your CICS Web support architecture for CICS as an HTTP server

The CICS Web support architecture for CICS as an HTTP server varies depending on the tasks you want it to perform. Some configurable components of CICS Web Support are required for all tasks, such as the TCPIPSERVICE definitions for the ports that receive inbound requests. Other configurable components are only required for specific tasks.

Before you begin

Before you start to plan your CICS Web support architecture for CICS as an HTTP server, read the topic "HTTP request and response processing for CICS as an HTTP server" on page 26 so that you understand the processing stages that can be involved.

Procedure

- You can serve dynamic, application-generated responses to HTTP requests, using a specially designed Web-aware CICS application program. "Providing dynamic HTTP responses with Web-aware application programs" tells you how to do this.
- You can serve static responses to HTTP requests, using documents or files that are available to CICS. "Providing static HTTP responses with a CICS document template or z/OS UNIX file" on page 64 tells you how to do this.
- You can enable a Web client using HTTP to access an existing COMMAREA application in CICS. "Giving Web clients access to COMMAREA applications" on page 68 tells you how to do this.
- You can enable a Web client using HTTP to access an existing 3270 display application in CICS. Chapter 15, "CICS Web support and 3270 display applications," on page 193 tells you how to do this.
- You can receive non-HTTP requests from a client and provide an application-generated response. Chapter 14, "CICS Web support and non-HTTP requests," on page 187 tells you how to do this.

Providing dynamic HTTP responses with Web-aware application programs

You can use Web-aware application programs to provide application-generated responses to HTTP requests from a Web client.

Before you begin

The base components of CICS Web support must be configured before you start, as described in Chapter 4, "Configuring CICS Web support base components," on page 53.

About this task

The following task-specific components of CICS Web support are used for this task:

- TCPIPSERVICE resource definitions
- URIMAP resource definitions
- Web-aware application programs, using the EXEC CICS WEB programming interface
- Alias transactions for the application programs
- Analyzer program
- Security facilities
- Web error programs

Figure 4 on page 61 shows the architecture elements for this CICS Web support task. "HTTP request and response processing for CICS as an HTTP server" on page 26 explains how the process elements work together.

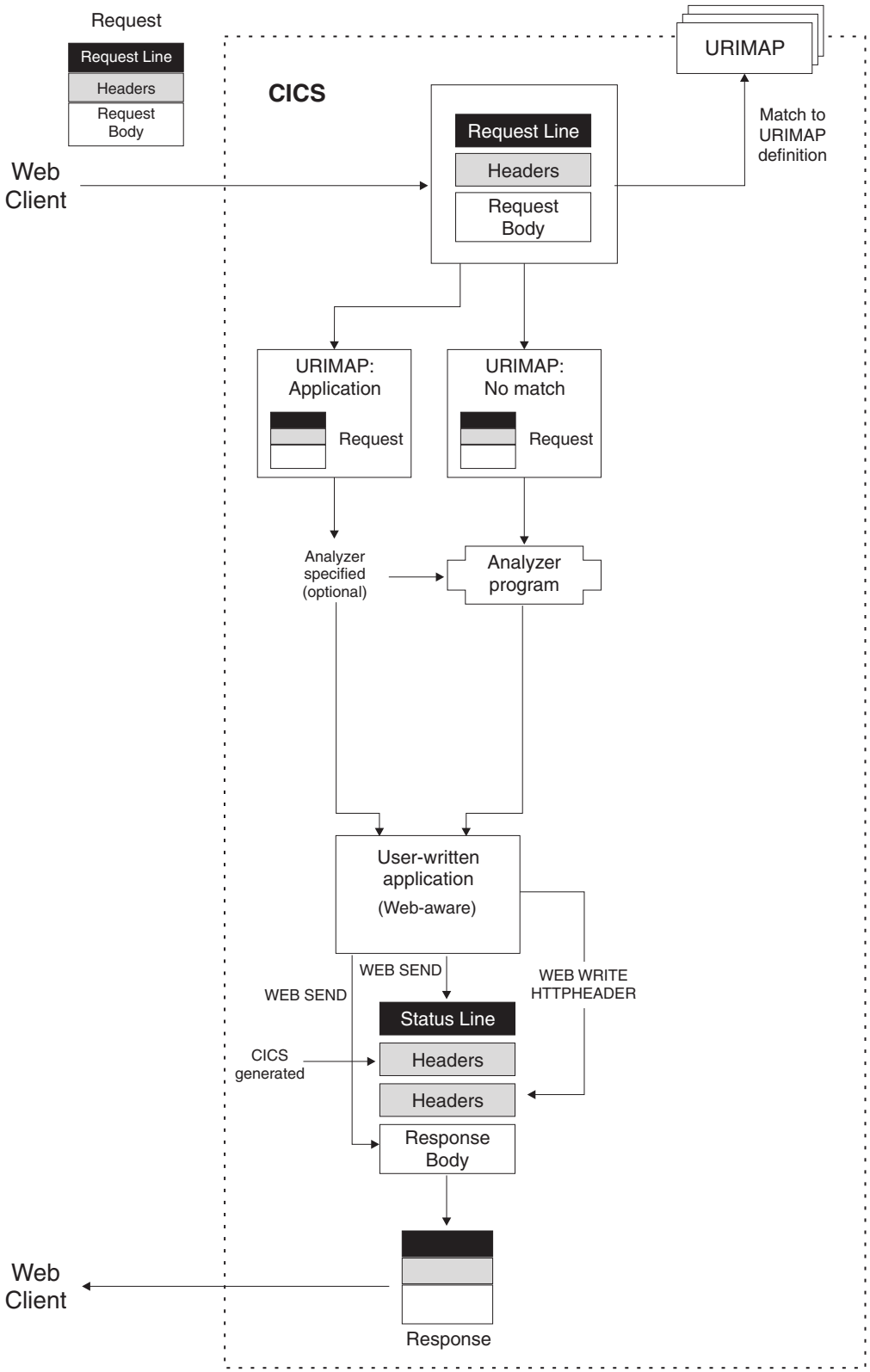


Figure 4. Dynamic HTTP responses with Web-aware application programs

Procedure

1. Design and code one or more Web-aware application programs to provide a response to each request by the Web client, using the information in

Chapter 6, “Writing Web-aware application programs for CICS as an HTTP server,” on page 75. Web-aware application programs can use EXEC CICS WEB and EXEC CICS DOCUMENT commands to receive and analyze an HTTP request, and to write and send a response to the request. Each program handles a single request and response.

Note: Web-aware application programs that use the EXEC CICS WEB commands must run in the CICS region where the Web client's request is received. However, they may link to application programs in other CICS regions, for example to perform business logic.

2. Consider security issues for this CICS Web support task. CICS can implement HTTP basic authentication for a connection, where the user must supply an ID and password. You can use the user ID to control access to individual resources used for application-generated responses or static responses. If you need to protect information passed over the Internet (including the user IDs and passwords used for basic authentication), consider using Secure Sockets Layer (SSL). For more information, see Chapter 13, “Security for CICS Web support,” on page 173.
3. Decide on the URL that the Web client will use for each request, including the scheme, host and path components, and any query string. “The components of a URL” on page 10 explains each of these components and how they are delimited. “URLs for CICS Web support” on page 33 explains the factors and limitations to consider in choosing a URL for CICS Web support.
4. Decide on the port that will be used for the requests. “Reserving ports for CICS Web support” on page 54 has more information about ports that can be used by CICS Web support. For HTTP, the default port number is 80, and for HTTPS (with SSL), the default port number is 443. Port numbers that are not the default for a scheme need to be specified explicitly in the URL of requests. If you prefer, you can allow a request to use any port that is associated with CICS Web support.
5. If you do not yet have a TCPIPSERVICE definition for the port on which the requests are received, follow the procedure in “Creating TCPIPSERVICE resource definitions for CICS Web support” on page 96 to set up a TCPIPSERVICE definition. Use this definition to specify the security measures that you have selected for the port (such as the use of SSL and basic authentication). The name of the relevant TCPIPSERVICE definition is specified in the URIMAP definition for the request. Specifying no TCPIPSERVICE definition means that requests matched by the URIMAP definition can use any port for which a TCPIPSERVICE definition exists.
6. Select an alias transaction ID for the user application programs that are involved with this task. The default alias transaction is CWBA. You can create your own alias transaction following the instructions in “Creating TRANSACTION resource definitions for CICS Web support” on page 99. You can use the URIMAP definition or an analyzer program to specify an alias transaction for each HTTP request. If you are implementing resource level security using the user IDs of Web clients, the user IDs are applied to this transaction, and need permission to access protected CICS resources and commands used by the transaction.
7. Decide how the analyzer program associated with the TCPIPSERVICE definition should be used, and select an appropriate program. Chapter 10, “Analyzer programs,” on page 125 has more information about what you can do using an analyzer program. For Web-aware applications, you can choose between the following strategies:

a. Use the CICS-supplied default analyzer program DFHWBAAX to provide error handling. DFHWBAAX is suitable where all of the requests using this port are handled using URIMAP definitions. DFHWBAAX takes no action if a matching URIMAP definition is found. If no match is found, it gives control to the user-replaceable Web error program DFHWBERX to produce an error response.

b. Use the CICS-supplied sample analyzer program DFHWBADX to provide basic support for both requests using URIMAP definitions, and requests following the same process that CICS Web support used before CICS TS 3.1. DFHWBADX takes no action if a matching URIMAP definition is found. If no match is found, it analyzes URLs in the format that was required before CICS TS 3.1. If the analysis fails, DFHWBADX gives control to the user-replaceable Web error program DFHWBEP to produce an error response. (If the URLs specified in your URIMAP definitions do not fit the format that was required before CICS TS 3.1, you should customize DFHWBADX or DFHWBEP so that a more meaningful response is provided.)

c. Use your own analyzer program to provide customized support. This might include:

   - Making dynamic changes to processing for requests using URIMAP definitions.
   - Providing monitoring or audit actions during processing for requests.
   - Supporting requests following the same process that CICS Web support used before CICS TS 3.1.
   - Providing error responses using either or both of the user-replaceable Web error programs, DFHWBEP and DFHWBERX.

   A customized analyzer program can be specified using the ANALYZER(YES) attribute in a URIMAP definition, and is then involved in the processing path for requests. As supplied, DFHWBAAX and DFHWBADX take no action if they are invoked from a URIMAP definition.

8. Decide how you want code page conversion to take place, for the HTTP requests, and for the responses that the user application programs provide to them. Code page conversion typically consists of converting the Web client's request, made using an ASCII character set, into an EBCDIC code page for use by the application program; and then reversing this process to return the application program's output to the Web client. “Code page conversion for CICS Web support” on page 39 explains the processes for code page conversion. You can specify code page conversion settings in the **EXEC CICS** WEB API commands issued by a Web-aware application program.
9. Set up a URIMAP definition to handle each request. Follow the procedures in “Starting a URIMAP resource definition for any requests for CICS as an HTTP server” on page 100 and “Completing a URIMAP definition for an application response to an HTTP request for CICS as an HTTP server” on page 102. It is possible to pass HTTP requests directly to an analyzer program without using URIMAP definitions, following the same process that CICS Web support used before CICS TS 3.1. However, using URIMAP definitions can make it easier to manage HTTP requests. Without URIMAP definitions, if you want to change the way in which CICS responds to a particular HTTP request, you need to change the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system management task.
10. Ensure that the user-replaceable Web error programs which are involved in your architecture provide appropriate responses to the Web client. The Web

error programs are used if an error in initial processing, or an abend or failure situation, occurs in the CICS Web support processThey are not used in all error situations. Chapter 9, “Web error program,” on page 115 explains the situations in which DFHWBEP and DFHWBERX are used, and tells you how to customize the responses that they provide.

Providing static HTTP responses with a CICS document template or z/OS UNIX file

You can use a CICS document template or a z/OS UNIX System Services file to provide a static response to an HTTP request from a Web client.

Before you begin

The base components of CICS Web support must be configured before you start, as described in Chapter 4, “Configuring CICS Web support base components,” on page 53.

About this task

The following task-specific components of CICS Web support are used for this task:

- TCPIPSERVICE resource definitions
- URIMAP resource definitions
- z/OS UNIX files
- CICS document template support
- Security facilities
- Web error programs

Figure 5 on page 65 shows the architecture elements for this CICS Web support task. “HTTP request and response processing for CICS as an HTTP server” on page 26 explains how the process elements work together.

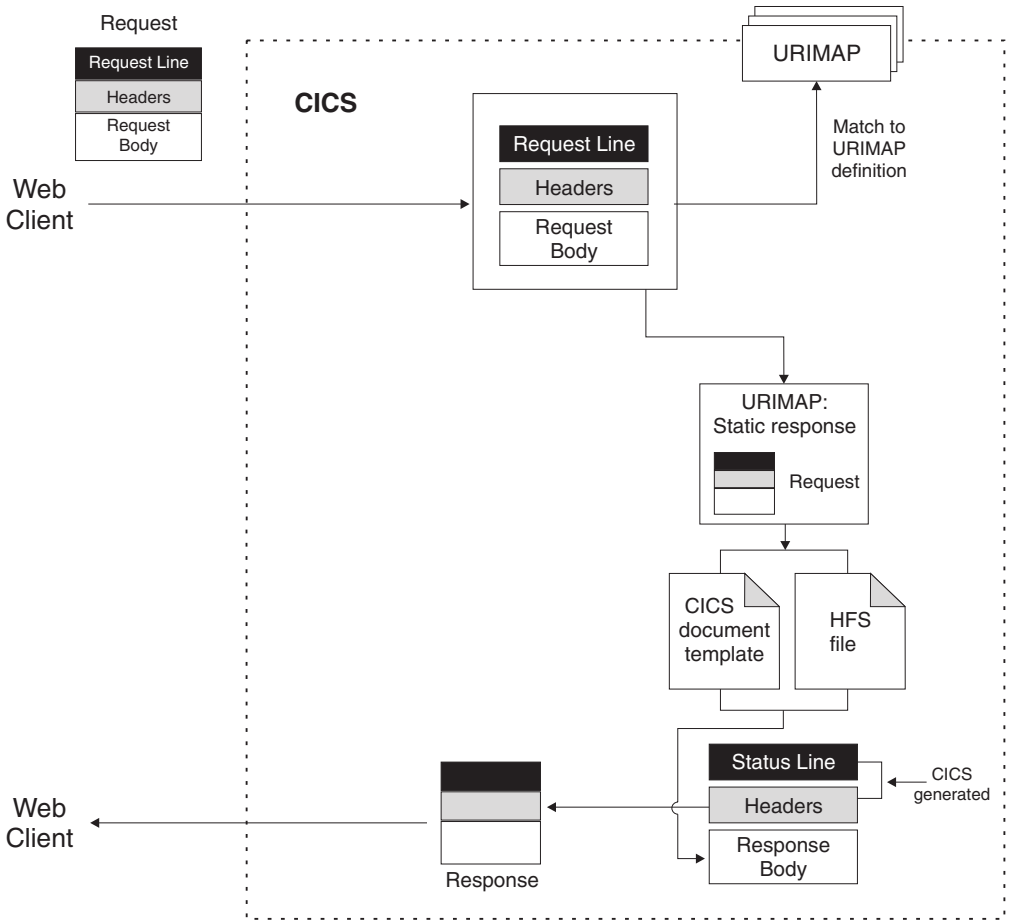


Figure 5. Static HTTP responses

Procedure

1. Consider security issues for this CICS Web support task. CICS can implement HTTP basic authentication for a connection, where the user must supply an ID and password. You can use the user ID to control access to individual resources used for application-generated responses or static responses. If you need to protect information passed over the Internet (including the user IDs and passwords used for basic authentication), consider using Secure Sockets Layer (SSL). For more information, see Chapter 13, “Security for CICS Web support,” on page 173.
2. If you want to use a z/OS UNIX System Services file to provide a response, create the file and place it in an appropriate location of your choice in the z/OS UNIX file system. When this response is identified by a URIMAP definition that matches the Web client's request, CICS retrieves the file and carries out appropriate code page conversion.
   a. Do not include any HTTP headers or status line information in the z/OS UNIX file. CICS generates the required information when the response is sent. The z/OS UNIX file only provides the body of the response.
   b. The location of the file is significant if you want to use path matching, as described later in this topic. If you do not want to use path matching, the location of the file does not need to have any relationship to the URL of the request.

c. The CICS region must have permissions to access z/OS UNIX, and it must have permission to access the directory containing the file, and the file itself. *Java Applications in CICS* explains how to grant these permissions.

d. If you are implementing access control using the user IDs of Web clients, the user IDs must also have permission to access the directory containing the file, and the file itself. "CICS system and resource security for CICS Web support" on page 178 explains more about this.

3. If you want to use a CICS document template to provide the response, create the document template, following the instructions in the *CICS Application Programming Guide*. The document template is defined using a DOCTEMPLATE resource definition. The document template can be held in a partitioned data set, a CICS program, a file, a temporary storage queue, a transient data queue, an exit program or a z/OS UNIX System Services file. When this response is identified by a URIMAP definition that matches the Web client's request, CICS creates a document using the template, retrieves the document, and carries out appropriate code page conversion.

   a. Do not include any HTTP headers or status line information in the document template. CICS generates the required information when the response is sent. The document template only provides the body of the response.

   b. A query string that consists of name and value pairs can be used as a symbol list and substituted into a document template. (The query string cannot be used in this way if it has already been used for URIMAP matching, as part of the PATH attribute in the URIMAP definition.) The recommended way to get the client to send a query string of the expected format in the URL, is to send an HTML form with the GET method for the user to fill in. Any of the names in the query string can be coded in the document template as a symbol, and when the template is used, CICS substitutes each symbol for the value specified in the query string. For example, if you have obtained a query string that includes a name and value pair `username=Peter`, you can use this in your document template by coding `username` as a symbol:

      ```
      Welcome to the finance system, &username;.
      ```

      The resulting static response delivered to the user will read

      ```
      Welcome to the finance system, Peter.
      ```

      Note: Symbols in document templates are case-sensitive. Specify the name using the same case as is present in the original query string.
      Any name and value pairs that do not correspond to symbols in the document template are ignored.

   c. If you are implementing resource level security using the user IDs of Web clients, the user IDs must have permission to access the document template. "CICS system and resource security for CICS Web support" on page 178 explains how to grant this. Note that if the document template is a z/OS UNIX System Services file, the Web clients do not need to be given permissions on the file, but only on the DOCTEMPLATE resource definition.

4. Identify the media type (type of data content) that is provided by the z/OS UNIX file or CICS document template. See "IANA media types and character sets" on page 10 for more information about media types. Note that when you use a URIMAP definition to send a static response, the use of quality factors (the "q" parameter) is not supported. Quality factors can be used to choose among a client's list of acceptable media types or character sets, as specified in

Accept headers. If you want to carry out this type of analysis, an application-generated response can be used instead.

5. Identify the information that CICS requires for code page conversion of the static response. Code page conversion only takes place where a text media type is specified. "Code page conversion for CICS Web support" on page 39 explains the processes for code page conversion.
   a. Identify the character set into which CICS should convert the static response before sending it to the Web client. The IANA character sets supported by CICS for code page conversion are listed in Appendix A, "HTML coded character sets," on page 373.
   b. Identify the IBM code page (EBCDIC) in which the document template or z/OS UNIX file providing the response body is encoded.

   For a static response, this information is specified in the URIMAP definition for the request.
6. Decide on the URL that the Web client will use for each request, including the scheme, host and path components, and any query string. "The components of a URL" on page 10 explains each of these components and how they are delimited. "URLs for CICS Web support" on page 33 explains the factors and limitations to consider in choosing a URL for CICS Web support.
7. Decide whether you want to use path matching in the URIMAP definition. If so, plan your request URLs, and arrange the names of your CICS document templates or the locations of your z/OS UNIX files to support this. In path matching, a wildcard character is used in the path component of the URIMAP definition, and also in the name of the CICS document template or z/OS UNIX file that is specified by the URIMAP definition. The portion of the path that is covered by the wildcard character is used to select the document template or z/OS UNIX file to provide the response.
   a. For CICS document templates, the portion of the path that is covered by the wildcard character is substituted as the last part of the template name. You can create a collection of document templates whose names begin in the same way, and access them using request URLs whose paths begin in the same way, through a single URIMAP definition.
   b. For z/OS UNIX files, the portion of the path that is covered by the wildcard character is substituted as the last part of the file name. You can store a number of these files in the same directory, and access them using request URLs whose paths begin in the same way, through a single URIMAP definition. Bear in mind that because a URIMAP definition must specify a type of data content (the MEDIATYPE attribute), a single URIMAP definition can only handle a group of z/OS UNIX files that produce the same type of data content.
8. Decide on the port that will be used for the requests. "Reserving ports for CICS Web support" on page 54 has more information about ports that can be used by CICS Web support. For HTTP, the default port number is 80, and for HTTPS (with SSL), the default port number is 443. Port numbers that are not the default for a scheme need to be specified explicitly in the URL of requests. If you prefer, you can allow a request to use any port that is associated with CICS Web support.
9. If you do not yet have a TCPIPSERVICE definition for the port on which the requests are received, follow the procedure in "Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 to set up a TCPIPSERVICE definition. Use this definition to specify the security measures that you have selected for the port (such as the use of SSL and basic authentication). The name of the relevant TCPIPSERVICE definition is

specified in the URIMAP definition for the request. Specifying no TCPIPSERVICE definition means that requests matched by the URIMAP definition can use any port for which a TCPIPSERVICE definition exists.

10. Set up a URIMAP definition to handle each request. Follow the procedures in "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100 and "Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server" on page 103. The URIMAP definition can identify either a z/OS UNIX file or a document template.
11. Check the error handling procedures for this CICS Web support task.
    a. Check the behavior of the analyzer program associated with the TCPIPSERVICE definition for the port on which the requests are received. If URIMAP matching fails for a request, the request is passed on to the analyzer program. If the port is used only for static responses, the CICS-supplied default analyzer program DFHWBAAX provides suitable error handling. Otherwise, the choice of analyzer program is likely to depend on the requirements of user application programs, and you might need to customize it to provide suitable error handling for static responses. Chapter 10, "Analyzer programs," on page 125 has more information about what you can do using an analyzer program.
    b. Ensure that the user-replaceable Web error programs which are involved in your architecture provide appropriate responses to the Web client. Chapter 9, "Web error program," on page 115 explains the situations in which DFHWBEP and DFHWBERX are used, and tells you how to customize the responses that they provide.

Related information:

Giving CICS regions permission to access HFS directories and files

CICS Web support resources on z/OS UNIX

When you use z/OS UNIX files to provide static responses to requests from Web clients, a CICS region which receives those requests and provides the responses needs **read** access to the files and to the directories containing them.

If you have stored all the files relevant to each CICS region in a directory structure below the home directory for the CICS region, you can make the CICS region the owner of each directory and file (with the appropriate owner permissions). If some z/OS UNIX files are used by more than one CICS region, you will need to use one of the other possible solutions, such as group permissions or access control lists (ACLs). The use of "other" permissions, which would give access to every z/OS UNIX user, is not recommended for CICS Web support in a production environment.

Related information:

Giving CICS regions permission to access HFS directories and files

Giving Web clients access to COMMAREA applications

You can use CICS Web support to enable Web clients to interact with CICS applications that use a COMMAREA interface to communicate with other programs. You can write a Web-aware application program that links to the application and uses its output to provide HTTP responses. Alternatively, you can use a converter program to convert the input from the Web client into a suitable COMMAREA, and convert the output from the application into an HTTP response.

Before you begin

The base components of CICS Web support must be configured before you start, as described in Chapter 4, "Configuring CICS Web support base components," on page 53.

About this task

The following task-specific components of CICS Web support are used for this task:

- TCPIPSERVICE resource definitions
- URIMAP resource definitions
- COMMAREA application programs
- Either: Web-aware application programs, using the **EXEC CICS** WEB programming interface, that link to the COMMAREA application programs and use their output
- Or: Converter programs that can provide suitable COMMAREA input and convert the output from the application programs into an HTTP response
- Alias transactions to cover the user application programs involved in this process
- Analyzer program
- Security facilities
- Web error programs

Figure 6 on page 70 shows the architecture elements for this CICS Web support task. "HTTP request and response processing for CICS as an HTTP server" on page 26 explains how the process elements work together.

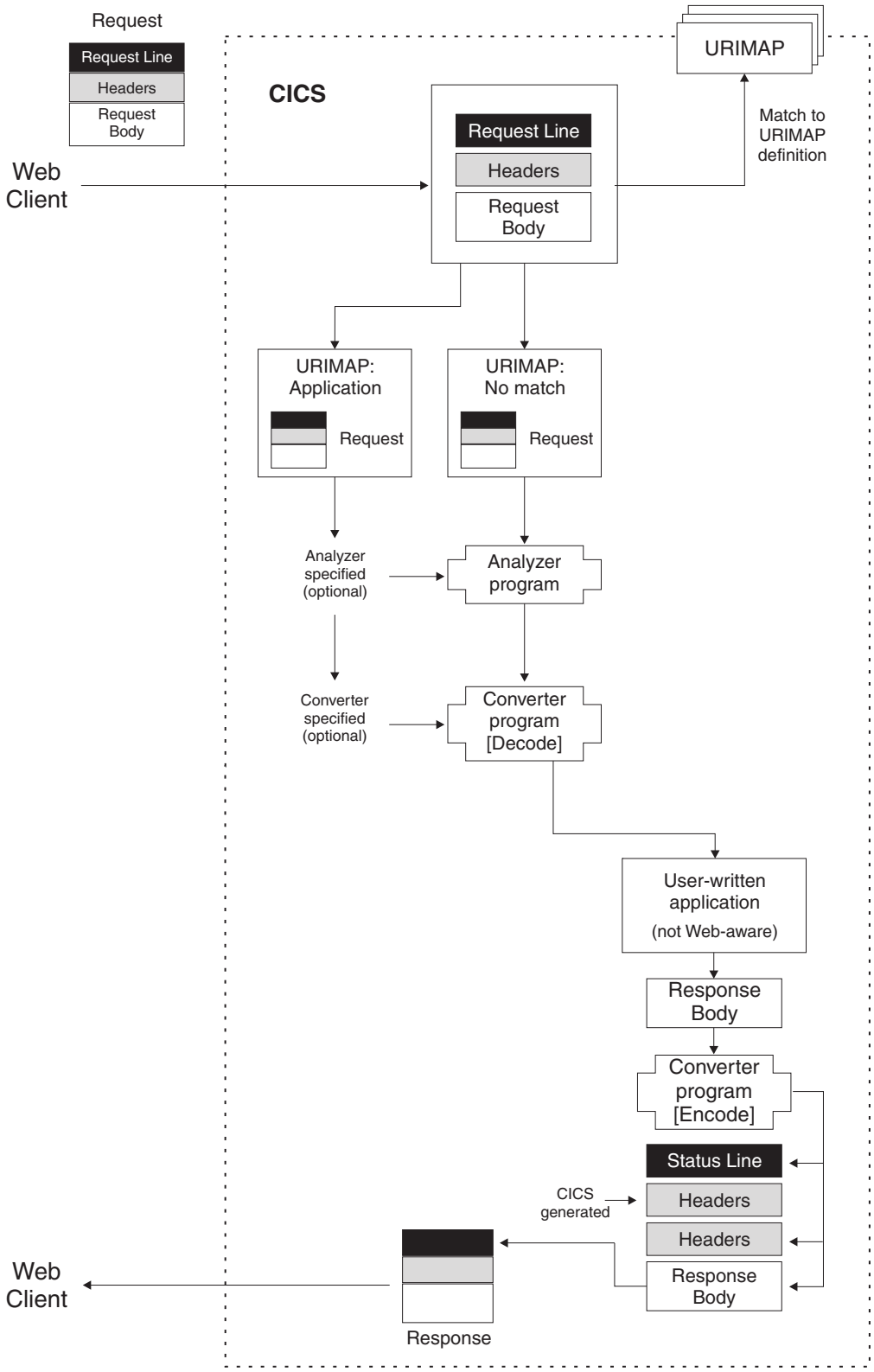


Figure 6. HTTP responses from a COMMAREA application

Procedure

1. Decide whether you want to write a Web-aware application program that handles the HTTP request and links to the COMMAREA application program;

or whether you want to write a converter program to convert the input from the Web client into a suitable COMMAREA, and convert the output from the application into an HTTP response. Converter programs can either use the **EXEC CICS** WEB API commands to read the Web client's request and produce the response, or they can work with the request and response in blocks of storage.

a. If you want to use a Web-aware application program, follow the steps in "Providing dynamic HTTP responses with Web-aware application programs" on page 59. Code your Web-aware application program to link to the COMMAREA application and use its output. The only task that cannot be performed by a Web-aware application program, but can be performed by a converter program, is receiving information that an analyzer program has created to pass to the next processing stage (in a user token or shared work area). This is unlikely to be a consideration when you are developing a new CICS Web support application.
b. If you want to use a converter program, follow the steps in this topic.

2. Consider security issues for this CICS Web support task. CICS can implement HTTP basic authentication for a connection, where the user must supply an ID and password. You can use the user ID to control access to individual resources used for application-generated responses or static responses. If you need to protect information passed over the Internet (including the user IDs and passwords used for basic authentication), consider using Secure Sockets Layer (SSL). For more information, see Chapter 13, "Security for CICS Web support," on page 173.
3. Decide how the analyzer program associated with the TCPIPSERVICE definition should be used, and select an appropriate program. Chapter 10, "Analyzer programs," on page 125 has more information about what you can do using an analyzer program. URIMAP definitions or analyzer programs can be used to map requests from Web clients to appropriate converter programs and user-written application programs. For non-Web-aware applications, even if you have URIMAP definitions, you need to use a customized analyzer program in the processing path for the request in the following circumstances:
   a. If the converter program is working with the request and response in blocks of storage, and you require nonstandard code page conversion. Converter programs do not have a mechanism for specifying code page conversion for blocks of storage containing HTTP requests and responses. In the absence of an analyzer program, CICS uses the standard settings described in "Writing a converter program" on page 140 to convert the message body supplied in the block of storage on both input and output. If this behavior is not suitable, you either need to use an analyzer program in the processing path to specify alternative settings, or use the **EXEC CICS** WEB API commands instead of working with the blocks of storage."Code page conversion for CICS Web support" on page 39 has more information about the processes for code page conversion.
   b. If you need to communicate any information to a converter program in addition to the standard input. A user token is provided, which the analyzer and converter programs can use to exchange either a small amount of information, or the address of a shared work area containing more information.
   c. If you require monitoring or audit actions, which can be carried out by an analyzer program.
   d. If you need to make dynamic changes to elements of the process such as the converter program that is used, the application program that handles the request, or the alias transaction and user ID used for the request.

If you do not need any of these functions, you can use the CICS-supplied default analyzer program DFHWBAAX, or the CICS-supplied sample analyzer program DFHWBADX, to provide basic error handling. DFHWBAAX is suitable where all of the requests using this port are handled using URIMAP definitions. DFHWBADX provides basic support for both requests using URIMAP definitions, and requests following the same process that CICS Web support used before CICS TS 3.1.

4. Use the information in Chapter 11, "Converter programs," on page 139 to create a suitable converter program. The converter program is called twice, first for the **decode** function, which examines the Web client's request and any additional information supplied by the URIMAP definition or analyzer program, and creates a suitable COMMAREA to pass to the application program. Next, the converter program is called for the **encode** function, which receives the application program's output and creates an HTTP response. If more than one application program is to supply data, the converter program can call the decode function repeatedly. "Calling more than one application program from a converter program" on page 145 explains how this is achieved.
5. Decide on the URL that the Web client will use for each request, including the scheme, host and path components, and any query string. "The components of a URL" on page 10 explains each of these components and how they are delimited. "URLs for CICS Web support" on page 33 explains the factors and limitations to consider in choosing a URL for CICS Web support.
6. Decide on the port that will be used for the requests. "Reserving ports for CICS Web support" on page 54 has more information about ports that can be used by CICS Web support. For HTTP, the default port number is 80, and for HTTPS (with SSL), the default port number is 443. Port numbers that are not the default for a scheme need to be specified explicitly in the URL of requests. If you prefer, you can allow a request to use any port that is associated with CICS Web support.
7. Select an alias transaction ID for the user application programs that are involved with this task. The default alias transaction is CWBA. You can create your own alias transaction following the instructions in "Creating TRANSACTION resource definitions for CICS Web support" on page 99. You can use the URIMAP definition or an analyzer program to specify an alias transaction for each HTTP request. If you are implementing resource level security using the user IDs of Web clients, the user IDs are applied to this transaction, and need permission to access protected CICS resources and commands used by the transaction.
8. Set up a URIMAP definition to handle each request. Follow the procedures in "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100 and "Completing a URIMAP definition for an application response to an HTTP request for CICS as an HTTP server" on page 102. It is possible to pass HTTP requests directly to an analyzer program without using URIMAP definitions, following the same process that CICS Web support used before CICS TS 3.1. However, using URIMAP definitions can make it easier to manage HTTP requests. Without URIMAP definitions, if you want to change the way in which CICS responds to a particular HTTP request, you need to change the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system management task.
9. If you do not yet have a TCPIPSERVICE definition for the port on which the requests are received, follow the procedure in "Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 to set up a TCPIPSERVICE definition. Use this definition to specify the security measures

that you have selected for the port (such as the use of SSL and basic authentication). The name of the relevant TCPIPSERVICE definition is specified in the URIMAP definition for the request. Specifying no TCPIPSERVICE definition means that requests matched by the URIMAP definition can use any port for which a TCPIPSERVICE definition exists.

10. Check the error handling procedures for this CICS Web support task.
    a. Check the behavior of the analyzer program associated with the TCPIPSERVICE definition for the port on which the requests are received. If URIMAP matching fails for a request, the request is passed on to the analyzer program. Chapter 10, "Analyzer programs," on page 125 has more information about what you can do using an analyzer program.
    b. Ensure that the user-replaceable Web error programs which are involved in your architecture provide appropriate responses to the Web client. Chapter 9, "Web error program," on page 115 explains the situations in which DFHWBEP and DFHWBERX are used, and tells you how to customize the responses that they provide.

Chapter 6. Writing Web-aware application programs for CICS as an HTTP server

In CICS, Web-aware application programs are programs that use **EXEC CICS** WEB commands to interact with a Web client or a server through CICS. For CICS as an HTTP server, these programs can receive and analyze HTTP requests and provide application-generated responses to the Web client.

Before you begin

Before you start to code Web-aware application programs for CICS as an HTTP server, read the topic "HTTP request and response processing for CICS as an HTTP server" on page 26 so that you are aware of the processing stages that can be involved.

If you want the service you are providing to Web clients to be compliant with the HTTP protocol specifications, in particular HTTP/1.1, read the topic "HTTP/1.1 compliance for CICS as an HTTP server" on page 43 for more information about the actions that CICS and your user application can take to achieve this.

About this task

For each HTTP request that requires an application-generated response, CICS calls the Web-aware application program that is specified on the URIMAP definition for the request, or by the analyzer program, if an analyzer is used. If you use a URIMAP definition to specify the application program, you can select a single application program to service all requests using a particular URL. If you are using an analyzer program either instead of, or in addition to, the URIMAP definition, it can carry out analysis on the request and decide on an alternative application program.

Remember: Web-aware application programs that use the EXEC CICS WEB commands must run in the CICS region where the Web client's request is received. However, they may link to application programs in other CICS regions, for example to perform business logic.

For CICS as an HTTP server, when an application program has sent a response to a request and returned control to CICS, it does not wait for further requests from the Web client. This is the case even when requests form a logical sequence, or are made using a persistent connection, or are pipelined. If you need to share information between different programs (or new instances of the same program) across a series of requests and responses, you can do this using CICS-managed resources, or using elements of the requests sent by the Web client.

You can code each of your Web-aware application programs to perform some or all of the following actions for processing an HTTP request:

Procedure

1. Retrieve any information that your application program needs from the request line (including the request URL), using the WEB EXTRACT command. "Examining the request line for an HTTP request" on page 77 tells you how to do this. The request line includes the HTTP method, which indicates the action that the application program should take. You can also design the path

component of a request URL to provide processing information to the application program. If there is a query string in the request URL, the application program can extract it as a whole for processing.

2. Read or browse the HTTP headers for the request, using the HTTP header commands. "Examining the HTTP headers for a message" on page 78 tells you how to do this. The information in the HTTP headers might be useful to the application program for processing and responding to the request.
3. Retrieve any technical information about the request that your application program needs to use. You can use **EXEC CICS** commands to access information about the TCP/IP environment and security options. "Retrieving technical and security information about an HTTP request" on page 79 tells you how to do this.
4. If the request contains form data that you want to extract, read or browse the data using the form field commands. "Examining form data in an HTTP request" on page 80 tells you how to do this. The data can be in the body of the request or as a query string in the URL, and CICS can extract the data from either of these locations.
5. If the request has a message body that you need to use, receive it into a buffer using the WEB RECEIVE command. "Receiving the entity body of an HTTP request" on page 82 tells you how to do this. CICS does not require you to receive a message body if one is present, and some requests do not have a message body.
6. Execute the business logic for the request processing, using the information you have gathered. You might want to involve other application programs to perform processing. A Web-aware application program can produce a response to the HTTP request based on information that it receives from non-Web-aware programs. It is advisable to separate the business logic from the presentation logic. In a Web-aware application, presentation logic controls the interaction with the Web client. For advice on how to separate business and presentation logic, see the *CICS Application Programming Guide*.
7. Write HTTP headers for the response, using the WEB WRITE HTTPHEADER command. "Writing HTTP headers for a response" on page 84 tells you how to do this. CICS automatically provides some required headers, such as the Date header. You can provide additional headers for other purposes.
8. Produce an entity body, or message body, which is the content of the HTTP response. "Producing an entity body for an HTTP message" on page 86 tells you how to do this. The entity body can be formed from a CICS document (which is created using the **EXEC CICS** DOCUMENT application programming interface) or from a buffer of data supplied by the application program.
9. Send the response to the Web client using the WEB SEND command. "Sending an HTTP response from CICS as an HTTP server" on page 86 tells you how to do this. You need to select a suitable status code and reason phrase, and specify the entity body. CICS assembles the response using these items and the HTTP headers. If you want to use chunked transfer-coding, you also need to follow the special instructions in "Using chunked transfer-coding to send an HTTP request or response" on page 89.
10. If you expect to exchange further requests and responses with this Web client, and you need to share data across the request sequence, use the suggestions in "Managing application state across an HTTP request sequence" on page 91 to achieve this.

What to do next

When **EXEC CICS** WEB commands are used for CICS as an HTTP server, they do **not** have the SESSTOKEN option. The SESSTOKEN option indicates that a command is being used for CICS as an HTTP client.

Examining the request line for an HTTP request

CICS stores the request line used for each HTTP request, for the application program to access if needed. An application program can use the WEB EXTRACT command to extract components of the request URL (including the path, host name, port number and query string), the method used for the request, or the HTTP version of the request. Non-HTTP requests can also be identified in this way.

About this task

"HTTP requests" on page 12 explains the items in a request line. The request URL is a major element of the request line; "The components of a URL" on page 10 explains the different parts of a URL. Your application program might need to examine any of the items in the request line in order to process the request and provide an appropriate response. Some common reasons for extracting information from a request line are:

- Because the same application program is called to handle a number of different requests, perhaps as part of a logical request sequence, or as different requests that relate to the same resource.
- To see what action is being requested from the application by the HTTP method. Appendix D, "HTTP method reference for CICS Web support," on page 391 explains the different methods that a Web client might use for a request, and suggests action that is appropriate in each case.
- To use the path component of the URL. This identifies the resource to which the request applies. As well as being used to map the request to the handling application, the path component of the URL can be designed to provide processing information to the application. For example, the path component can be used to specify a particular function provided by the application. Or if the Web-aware application is providing a front end for more than one other application, the path component of the URL can identify the application to which the request applies."URLs for CICS Web support" on page 33 explains how this can be achieved.
- To obtain a query string for processing by the application.
- To identify the HTTP version for the Web client, so that the application can provide an appropriate response. The HTTP version used by the Web client can affect the HTTP headers, status code, and message content for the response. HTTP/1.0 clients might not understand the more advanced features described in the HTTP/1.1 specification.
- To identify a non-HTTP request. Chapter 14, "CICS Web support and non-HTTP requests," on page 187 has more information about handling non-HTTP requests.

The *CICS Application Programming Reference* has full reference information and descriptions of the options available on the WEB EXTRACT command. The WEB EXTRACT command lets you obtain the following items:

Procedure

- Use the HOST option to obtain the host component of the request URL, as specified either in the Host header field for the request, or in the request line (if the absolute URI form was used for the request).
- Use the HTTPMETHOD option to obtain the HTTP method for the request (for example, GET or PUT).
- Use the HTTPVERSION option to identify the HTTP version, HTTP/1.1 or HTTP/1.0.
- Use the PATH option to obtain the path component of the URL.
- Use the PORTNUMBER option to obtain the port number that applies to the URL. 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.
- Use the QUERYSTRING option to obtain the whole of the query string. The query string is returned in its escaped form, with `%xx` sequences to represent certain characters that could hinder correct parsing. See "Escaped and unescaped data" on page 16 for an explanation of this. Alternatively, if the query string includes form data as name and value pairs (for example, `account=40138025`), you can use the WEB READ FORMFIELD command to obtain this data in an unescaped form. "Examining form data in an HTTP request" on page 80 tells you how to do this.
- Use the REQUESTYPE option to identify a non-HTTP request.

Examining the HTTP headers for a message

Each HTTP header for a request or response message consists of a header name and header value. CICS stores this information for the application to access if required. An application can receive the value of a specified header, or browse through the names and values of all the headers for a request or response. You can also convert an architected date and time stamp string taken from a header into the ABSTIME format.

About this task

Your application might need to examine information in the headers in order to process a request or response, and construct subsequent messages. For example:

- The TE header tells the application whether or not trailing headers are permitted in a chunked response message.
- Conditional headers can provide instructions to the application, such as to reply only if the response document has changed.

Bear in mind that unless you know the exact format of the HTTP request or response, your application should not rely on the presence of any particular header, as Web clients and servers can be inconsistent in the headers they send.

Some HTTP headers contain date and time stamps. CICS provides the CONVERTTIME command to convert common formats for architected date and time stamp strings into the ABSTIME format, for use by the application.

The standard HTTP headers are described in the HTTP/1.1 specification (RFC 2616) and the HTTP/1.0 specification (RFC 1945). Appendix B, "HTTP header reference for CICS Web support," on page 375 explains the general use of HTTP

headers in CICS Web support, and the actions that CICS Web support takes for specific headers received on messages. Some HTTP headers are ignored by CICS, and it is up to the user application to take appropriate action in response to these. Check the HTTP specification to which you are working for detailed guidance and requirements about the meaning and correct use of each HTTP header.

If the message includes any trailing headers, you can read these using the **EXEC CICS** WEB commands in the same way as for standard headers. The Trailer header on the message specifies the names of all the HTTP headers that were sent as trailing headers.

Procedure

- To examine the contents of a particular HTTP header, use the WEB READ HTTPHEADER command. Your application program needs to provide a buffer that receives the contents of the header. CICS returns a NOTFND condition if the header is not present in the request.
- To browse all the headers in a request or response:
  1. Use the WEB STARTBROWSE HTTPHEADER command to begin browsing the header lines.
  2. Use the WEB READNEXT HTTPHEADER command to retrieve the header name and header value for each line. Your application program needs to provide two buffers: one receives the name of the header, and one receives its contents. CICS returns an ENDFILE condition when all headers have been read.
  3. Use the WEB ENDBROWSE HTTPHEADER command to end the browse when your program has retrieved all the header information of interest.
- To convert an architected date and time stamp string that is provided in a HTTP header, receive it into a buffer using the WEB READ HTTPHEADER command, and then process it using the CONVERTTIME command. 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. These are RFC 1123 format (the Web standard), RFC 850 format (an older format), and ASCtime format (output from C function). The ABSTIME can be converted to other formats by the application, using the FORMATTIME command.

Retrieving technical and security information about an HTTP request

An application can obtain information about the TCP/IP environment for an HTTP request, including the security options that are in use, and about a client certificate that has been provided by a Web client.

About this task

CICS manages the TCP/IP connection between a Web client and server, applies appropriate security measures, and manages the process of authenticating the identity of a Web client. The actions taken by CICS for each connection are determined by the options you set in the TCPIPSERVICE definition for the port on which the Web client request is received. A user-written application can examine information obtained by this process, if this information is useful for determining how to process the request. For example, you can obtain the host name and IP address of the Web client that sent the HTTP request, or check the level of security and encryption for the connection.

The EXTRACT TCPIP command provides information about the TCP/IP connection, and about security options specified in the TCPIPSERVICE definition. The EXTRACT CERTIFICATE command provides information taken from any X.509 client certificate that was received from the Web client during a Secure Sockets Layer (SSL) handshake. The *CICS Application Programming Reference* has full reference information and descriptions of the options available on these commands.

Procedure

- To obtain the host name and IP address of the Web client that sent the HTTP request, use the EXTRACT TCPIP command with the CLIENTNAME and CLIENTADDR options. The IP address is available as a binary number, or as a character string containing its colon hexadecimal or dotted decimal representation.
- To obtain the host name and IP address of the host system on which the application is running (that is, CICS itself), use the EXTRACT TCPIP command with the SERVERNAME and SERVERADDR options. Again, the IP address is available as a binary number or as a character string containing its colon hexadecimal or dotted decimal representation.
- To obtain the number of the port on which the request was received, you can use the EXTRACT TCPIP command with the PORTNUMBER option. The port number is available as a binary number or a character string. Alternatively, you can use the WEB EXTRACT command with the PORTNUMBER option.
- To obtain the name of the TCPIPSERVICE resource definition associated with the request, use the EXTRACT TCPIP command with the TCPIPSERVICE option.
- To identify the type of authentication (basic authentication, client certificate authentication, or no authentication) that was specified in the TCPIPSERVICE definition, use the EXTRACT TCPIP command with the AUTHENTICATE option. "CICS as an HTTP server: authentication and identification" on page 173 explains more about the different types of authentication.
- To identify whether Secure Sockets Layer (SSL) support is specified in the TCPIPSERVICE definition, and the level of SSL encryption that is used, use the EXTRACT TCPIP command with the SSLTYPE and PRIVACY options. "SSL with CICS Web support" on page 184 explains more about SSL.
- To retrieve information from an X.509 certificate that was received from the Web client during an SSL handshake, use the EXTRACT CERTIFICATE command. CICS has already verified the supplied certificate by checking it against the security manager database and against a certificate revocation list that you can set up. A certificate contains fields that identify the subject (sometimes called the owner, or the user) of the certificate, and fields that identify the Certificate Authority that issued the certificate (the issuer). You can select the information that you require by specifying the OWNER or ISSUER option. You can also use the SERIALNUM and USERID options to retrieve the serial number of the certificate and the RACF® user ID associated with the certificate. The *CICS RACF Security Guide* explains more about the content of certificates and how they are used.

Examining form data in an HTTP request

Form data is information provided by the user through interaction with an element in a HTML form (such as a text input box, button, or check box). The information is transmitted as a series of name and value pairs. CICS can scan an HTTP request to pick out the form fields, so an application can obtain the data using CICS commands, without needing to receive and analyze the entire body of the request.

About this task

"HTML forms" on page 17 explains more about forms and form fields.

An application can receive the value of a specified form field, or browse through the names and values of all the form fields contained in a request. You can specify code page conversion options if you need to convert the data into a different code page for use by your application.

The Web client sends form data in a query string when the GET method is used, and in the message body when the POST method is used. CICS can extract the data from either of these locations, so you do not need to specify which method was used. As an alternative, if the form data is sent in the query string, you can retrieve the entire query string using the WEB EXTRACT command. "Examining the request line for an HTTP request" on page 77 tells you how to do that.

CICS only reads form data when CICS is the HTTP server. The facility is not available when CICS is an HTTP client.

Procedure

- To obtain the value of a particular field of an HTML form, use the WEB READ FORMFIELD command. Your application program can provide a buffer which will receive the value, or alternatively, you can provide a pointer which CICS sets to the address of the value. CICS returns a NOTFND condition if the form data does not contain a field with the specified name. The form data is unescaped by CICS before it is returned, with the %xx sequences converted back to the original characters. See "Escaped and unescaped data" on page 16 for an explanation of this.
- To browse all the fields in the form data:
  1. Use the WEB STARTBROWSE FORMFIELD command to begin browsing the fields.
  2. Use the WEB READNEXT FORMFIELD command to retrieve the name and value of each field in turn. Your application program needs to provide two buffers: one receives the name of the field, and one receives its contents. CICS returns an ENDFILE condition when all fields have been read.
  3. Use the WEB ENDBROWSE FORMFIELD command to end the browse when your program has retrieved all the fields of interest.
- CICS carries out code page conversion on the data you receive. You can use the CHARACTERSET and HOSTCODEPAGE options on the WEB STARTBROWSE FORMFIELD and WEB READ FORMFIELD commands to specify the code page used by the Web client and by your application program.
  1. The character encoding (charset parameter) used by a client application for both the GET and POST methods is determined by information in the HTML form. However, this information is not normally present as part of the submitted form request, so it is supplied by the application using the CHARACTERSET option. This information should match the forms encoding determined by the corresponding HTML form (see "How the client encoding is determined" on page 18 for more information).
  2. HOSTCODEPAGE specifies the CICS (host) code page used by the application program. This code page is normally an EBCDIC code page. If the code page is not specified, the data is returned in the EBCDIC code page specified by the LOCALCCSID system initialization parameter, provided that the specified code page is supported by the CICS web interface. Otherwise, CICS returns the data to the default EBCDIC code page 037.

For more information on the CHARACTERSET and HOSTCODEPAGE options, see the WEB READ FORMFIELD and WEB STARTBROWSE FORMFIELD commands.

Receiving the entity body of an HTTP request

An application can issue the WEB RECEIVE command to receive the entity body of an HTTP request. You can receive only the first part of the entity body, or use a series of WEB RECEIVE commands to receive the whole body in smaller sections.

About this task

The WEB RECEIVE command does not set a timeout value. The user application is only called when the complete request has been successfully received from the Web client, and is being held by CICS. For CICS as an HTTP server, the SOCKETCLOSE attribute in the TCPIPSERVICE definition for the port determines how long the Web client has to complete its request send. When this period expires, CICS returns a 408 (Request Timeout) response to the Web client.

If a request message is sent using chunked transfer-coding, CICS assembles the chunks into a single message before passing it to the application. If a series of pipelined requests is sent, CICS treats each request as a separate transaction, and requires a response from the user application before making the next request available to the next user application for processing.

The *CICS Application Programming Reference* has full reference information and descriptions of the options available on the WEB RECEIVE command. When you issue the WEB RECEIVE command:

Procedure

1. Identify whether or not you need to receive an entity body for this request.
   a. For certain request methods (such as the GET method), an entity body is not appropriate, and your application is allowed to ignore any entity body that is present. Appendix D, "HTTP method reference for CICS Web support," on page 391 indicates the methods where this applies. If an inappropriate entity body is present, you may still receive it if you want. "Examining the request line for an HTTP request" on page 77 tells you how to identify the request method.
   b. For an HTTP/1.1 request, the presence of an entity body is indicated by a non-zero Content-Length header on the request (or a Transfer-Encoding header, if the message is chunked). If the value of the Content-Length header is zero, or if neither the Transfer-Encoding header nor the Content-Length header is supplied, there is no entity body. "Examining the HTTP headers for a message" on page 78 tells you how to read the HTTP headers for the message.
   c. HTTP/1.0 requests are not required to specify a Content-Length header, but they might do so. If you find a non-zero Content-Length header on the request, this indicates the presence of an entity body. If there is no Content-Length header, but the request method (in particular, the POST method) indicates that an entity body is appropriate, it is likely that an entity body is present.
2. Receive the entity body by specifying either the INTO option (for a data buffer), or the **SET** option (for a pointer reference), and the LENGTH option. On return, the LENGTH option is set to the length of data received.

3. If you want to limit the amount of data received from the entity body, specify the MAXLENGTH option.
   a. If you want to receive only the first part of the entity body, and discard any data that exceeds this length, omit the NOTRUNCATE option. This is the default.
   b. If you want to retain, rather than discarding, any data that exceeds this length, specify the NOTRUNCATE option. Any remaining data can be obtained using further WEB RECEIVE commands.

   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 entity body for the chunked message, rather than to each individual chunk. The total amount of data that CICS accepts for a single message is limited by the MAXDATALEN attribute of the TCPIPSERVICE definition.
4. Specify any options that you want to set here for code page conversion.
   a. The SERVERCONV option provides overall control of code page conversion. Use it to specify whether or not code page conversion takes place. For CICS as an HTTP server, for compatibility with Web-aware applications coded in earlier releases, code page conversion is assumed if SERVERCONV is not specified but another code page conversion option is specified. If you want to prevent code page conversion, either specify SERVERCONV(NOSRVCONVERT), or omit all the code page conversion options.

      Note: If you receive an entity body that has been zipped or compressed, as indicated by a Content-Encoding header on the message, make sure that you suppress code page conversion. CICS does not decode these types of message for you, and if code page conversion is applied the results could be unpredictable. If you cannot decipher a zipped or compressed entity body, you can inform the Web client of this by returning a 415 status code.
   b. If you want code page conversion, but CICS cannot determine the Web client's character set, use the CHARACTERSET option to specify it. For older Web clients, the request headers might not provide this information. In this case, CICS assumes the ISO-8859-1 character set, so you only need to specify the character set if that assumption is not correct.
   c. If you want code page conversion, but the default code page for the local CICS region (as specified in the LOCALCCSID system initialization parameter) is not suitable for your application, use the HOSTCODEPAGE option to specify an alternative host code page.

   Code page conversion does not take place for messages that specify a non-text media type (unless you do not specify SERVERCONV, in which case for compatibility purposes, the media type is not taken into account). Note that for compatibility purposes, CICS deviates from the HTTP/1.1 requirement to default to `application/octet-stream` if inbound messages do not specify a media type. CICS uses `text/plain` as the default instead, so that code page conversion can be carried out for the message.
5. If you specified MAXLENGTH and NOTRUNCATE, and you have more data to receive, issue further WEB RECEIVE commands. 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. Bear in mind that if you

receive less than the length requested on the MAXLENGTH option, this does not necessarily indicate the end of the data; this could happen if CICS needs to avoid returning a partial character at the end of the data.

Writing HTTP headers for a response

For dynamic responses created by application programs, CICS automatically provides the HTTP headers that are required for basic messages, depending on the HTTP protocol version used for the message. You might need to add further HTTP headers to your response.

About this task

Some HTTP headers are created automatically by CICS if the message requires them. Your application does not need to write these headers. The full list of headers created by CICS is:

- ARM correlator
- Connection
- Content-Type (written by CICS, but can be supplied by a client application if a complex header is required)
- Content-Length
- Date
- Expect
- Host
- Server
- TE (written by CICS but further instances may be added)
- Transfer-Encoding
- User-Agent
- WWW-Authenticate

Note that some of these headers are appropriate, and created, only when CICS is an HTTP client. The circumstances in which these headers are created are described in Appendix B, "HTTP header reference for CICS Web support," on page 375. If you do write these headers on a response, CICS does not overwrite them, but uses the versions provided by your application.

The headers that CICS provides when a response is sent are the ones that should normally be written for a basic message to be compliant with the appropriate HTTP protocol specification. You might want to add further HTTP headers to the response for purposes such as:

- Control of caching and document expiry (for example, Cache-Control, Expires, Last-Modified)
- Content negotiation (for example, Accept-Ranges, Vary)
- Information for the Web client (for example, Title, Warning, further Content headers)

If your application program is performing complex actions, or if you select certain status codes for your response, the HTTP specification to which you are working is likely to require the use of particular HTTP headers for your message. When you add any HTTP headers to a response, check the HTTP specification to which you are working for any important requirements that apply to those headers. See "The HTTP protocol" on page 12 for more information about the HTTP specifications.

Write additional HTTP headers for a message **before** you issue the WEB SEND command to send the message. The exception to this rule is if you are writing headers to be sent as trailing headers on a chunked message, in which case the special process mentioned below applies. When writing HTTP headers for a response:

Procedure

- Use the WEB WRITE HTTPHEADER command for each header that you want to add to the message. Make sure that you specify the name and value for each header in the format described by the HTTP specification to which you are working. (CICS does not validate the content of HTTP headers, because you might want to use new or user-defined headers.) The command adds a single header, and you can repeat the command to add further headers. If you write a header that you have already written, CICS adds the new header to the request or response in addition to the existing header. Make sure that you only do this where the HTTP specification states that the header may be repeated. CICS stores the headers ready to be added to the request when it is sent.
- If any of the HTTP headers that you use might be unsuitable for Web clients below HTTP/1.1 level, before writing those headers, check the HTTP version information that the Web client has supplied to you. Use the WEB EXTRACT command to obtain this information. To allow you to use user-defined (nonstandard) headers, CICS does not remove unsuitable user-written headers. Some HTTP headers are not understood by servers below HTTP/1.1, and could lead to errors in processing your request.

  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 HTTP/1.0 as the HTTP version.
- If you want to produce a date and time stamp for use in one of your HTTP headers (for example, the Last-Modified header), you can use the FORMATTIME command. The STRINGFORMAT option on the command converts the current date and time (in ABSTIME format), or a date and time produced by the application program, into suitable date and time stamp formats for use on the Web. Other date and time stamp formats might not be accepted by some Web clients or servers with which CICS is communicating.
- If you want to produce a strong entity tag for use in the ETag HTTP header, you can use the SHA-1 digest produced by the BIF DIGEST command. The presence of a strong entity tag enables a client to make conditional requests for a resource using the entity tag in If-Match, If-None-Match, or If-Range headers, which is a more precise method of checking the status of a resource than the Last-Modified date and time string. If you want to allow conditional requests, your application program must provide support for them; CICS does not provide its own support for If-Match, If-None-Match, or If-Range functions on HTTP GET requests.
- If you are using chunked transfer-coding to send an HTTP request or response, and you want to include trailing headers at the end of the chunked message, follow the special instructions in “Using chunked transfer-coding to send an HTTP request or response” on page 89. You need to write a Trailer header before sending the first chunk of the message. All the HTTP headers written after the WEB SEND command for the first chunk are treated as trailing headers.
- Make sure that your application program carries out any actions that are implied by your user-written headers. For example, if you have written content-negotiation headers, your application program needs to provide different versions of the resource.

Producing an entity body for an HTTP message

Web-aware application programs can produce an entity body formed from a CICS document, or from a buffer of data.

Before you begin

About this task

CICS documents can be used as the entity body of an HTTP message. They are created using the **EXEC CICS** DOCUMENT commands. They can be populated by data specified directly by the application program, and by document templates, which are portions of documents defined as CICS resources or created by another CICS program. Documents and document templates can be stored for reuse.

Alternatively, you can specify a buffer of data created by the application program. You might find this option more convenient for short or simple entity bodies, and it is the only option that enables you to use chunked transfer-coding for the message. However, data created in this way cannot be stored for reuse so easily.

Procedure

1. To create a CICS document, follow the instructions in the *CICS Application Programming Guide*. The document is created using the **EXEC CICS** DOCUMENT application programming interface (**EXEC CICS** DOCUMENT CREATE, INSERT, and **SET** commands). The DOCTOKEN option on the WEB SEND command is used to specify the document token for the finished document. CICS retrieves the document and performs appropriate code page conversion, depending on the options you specify on the WEB SEND command. The body of a chunked message cannot be formed from CICS documents.
2. Alternatively, assemble a message body within your application program. The FROM option on the WEB SEND command is used to specify the buffer of data. There is no set maximum limit for the size of the data buffer, but you need to consider the following factors that could limit its size in practice:
   - The EDSA limit for the CICS region.
   - The number of other message bodies that you might be assembling at the same time in the CICS region. Scheduling constraints might be imposed by the MAXACTIVE setting for any transaction class definitions that apply to CICS Web support transactions.
   - The type of code page conversion used for the message body. For conversion from the EBCDIC code page 037 to the ASCII code page ISO-8859-1, CICS overwrites the same buffer of data, so no additional storage is used. For any other type of code page conversion, CICS requires additional storage to contain the converted message body. Depending on the character sets used, the size of this additional storage area can range from the same size as the original message body, to a theoretical maximum of four times the size of the original message body (which is unlikely). For example, a 2MB buffer of data sent using the FROM option would require at least 4MB of storage in total. Double-byte character sets (DBCS) or multi-byte character sets are likely to require larger storage areas within this range.

Sending an HTTP response from CICS as an HTTP server

Use the WEB SEND command to make CICS assemble the HTTP headers, entity body, status code and reason phrase for an HTTP response, carry out code page conversion, and transmit the response to the Web client.

Before you begin

Write any additional HTTP headers for the response using the WEB WRITE HTTPHEADER command before issuing the WEB SEND command, as described in "Writing HTTP headers for a response" on page 84. Also produce any entity body that is needed for the message, as described in "Producing an entity body for an HTTP message" on page 86.

You need to specify a status code and reason phrase on the WEB SEND command. "Status codes and reason phrases" on page 15 explains what these are. Appendix C, "HTTP status code reference for CICS Web support," on page 381 provides an overview of the status codes that your application might use. To plan your use of status codes and find further information about them, you should consult the HTTP specification to which you are working. See "The HTTP protocol" on page 12 for more information about the HTTP specifications.

About this task

If wanted, the response can be sent in chunks (chunked transfer-coding). You cannot send pipelined responses back to a Web client (you must send a single response to each request sent by the Web client).

The *CICS Application Programming Reference* has full reference information and descriptions of the options available on the WEB SEND command. When you issue the command:

Procedure

1. Specify the STATUSCODE option to select an appropriate status code for the response, depending on the situation, and the STATUSTEXT and STATUSLEN options to provide the reason phrase. CICS does not validate your choice of status code, and it is the user application's responsibility to ensure that the value is valid and conforms to the rules for HTTP status codes. Depending on the status code that you select, you might need to complete some or all of the following steps before issuing the WEB SEND command:
   a. Check the HTTP version of the Web client's request, to ensure that the status code can be understood. The HTTP/1.1 specification includes more status codes than the HTTP/1.0 specification.
   b. If the HTTP specification states that the status code should be accompanied by certain HTTP headers, use the WRITE HTTPHEADER command to create those headers.
   c. If the HTTP specification states that the status code should be accompanied by a message body giving special information, create an appropriate entity body. This is normally the case when the status code indicates an error or requests further action from the client. Message bodies are not allowed with status codes 204, 205, and 304. If you have selected a status code that does not allow a message body, and attempt to use a message body, CICS gives an error response to the WEB SEND command.
2. Identify the source of any entity body for the response by specifying either the DOCTOKEN option, for a CICS document that you have created, or the FROM option, for a body of data that you have assembled. If you are using the FROM option, specify the FROMLENGTH option to give the length of the entity body, or of the chunk if chunked transfer-coding is in use. For chunked transfer-coding, the DOCTOKEN option cannot be used.

3. Specify the media type for the body of the response, using the MEDIATYPE option. CICS does not check the validity of the specification against the data content. There is no default. If you do not specify this, CICS does not build a Content-Type header for the response.
4. If you want the message to be sent immediately, rather than at the end of the task (which is the default), specify IMMEDIATE for the ACTION option. If you are using chunked transfer-coding, IMMEDIATE is the default, so there is no need to make this choice.

   Note: 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.
5. If you want to close the connection after sending the response, specify CLOSE for the CONNECTION option. CICS writes a Connection: close header on the response, which notifies the Web client that the connection is closed and no more requests should be sent. (For a Web client at HTTP/1.0 level, CICS achieves the same effect by omitting the Connection: Keep-Alive header.)
6. Specify the appropriate settings for code page conversion of the message body.
   a. The SERVERCONV option provides overall control of code page conversion. Use it to specify whether or not code page conversion takes place. For CICS as an HTTP server, for compatibility with Web-aware applications coded in earlier releases, code page conversion is assumed if SERVERCONV is not specified but another code page conversion option is specified. If you want to prevent code page conversion, either specify SERVERCONV(NOSRVCONVERT), or omit all the code page conversion options.
   b. If you want code page conversion, but the character set selected by CICS is not suitable, use the CHARACTERSET option to specify an alternative. By default, CICS uses the character set specified in the Content-Type header of the Web client's original request. If that character set was unsupported or not stated, CICS uses the ISO-8859-1 character set instead.

      A Web client might specify alternative acceptable character sets in an Accept-Charset header. If you want to specify one of these, it is up to your application to analyze the header (which might include quality values to indicate the Web client's preference), and select an appropriate supported character set. CICS does not support all the character sets named by IANA. Appendix A, “HTML coded character sets,” on page 373 lists the IANA character sets that are supported by CICS for code page conversion.
   c. If you want code page conversion, and are using the FROM option to specify the message body, you need to use the HOSTCODEPAGE option to identify your application's code page, if this is **not** the default code page for the local CICS region (as specified in the LOCALCCSID system initialization parameter). If you are using a CICS document (DOCTOKEN option), CICS identifies the host code page from the CICS document domain's record of the host code pages for the document.

   Code page conversion does not take place for messages that specify a non-text media type (unless you do not specify SERVERCONV, in which case for compatibility purposes, the media type is not taken into account). The HTTP headers and status line are always converted into the ISO-8859-1 character set by CICS.
7. If you are using chunked transfer-coding (or chunking), in addition to the basic instructions in this topic, you need to follow the special instructions in “Using chunked transfer-coding to send an HTTP request or response” on page 89. You

need to ensure that the procedure described in that topic is followed correctly, so that the chunked message is acceptable to the recipient. Chunked messages are sent using several instances of the WEB SEND command, with particular options.

Using chunked transfer-coding to send an HTTP request or response

This topic tells you how to set up chunked transfer-coding for an HTTP request by CICS as an HTTP client, or an HTTP response from CICS as an HTTP server.

Before you begin

Before setting up chunked transfer-coding, you need to plan the following attributes of the item that you want to send:

1. The HTTP headers that should be used at the beginning of the message. CICS supplies its usual message headers, which are listed in Appendix B, "HTTP header reference for CICS Web support," on page 375. For a chunked message, CICS supplies the proper headers for chunked transfer-coding, including the Transfer-Encoding: chunked header. If any additional headers are required at the beginning of the message, the application can write them before the first WEB SEND command.
2. Any headers that should be sent in the trailer at the end of the message. These headers are known as trailing headers. Note that the HTTP/1.1 specification sets requirements for the use of trailing headers, including that it should not matter if the recipient ignores them.
3. How the message should be divided up. This can be done in whatever way is most convenient for the application program. For example, the output from a number of other application programs could be sent as it is produced, or data from each row of a table could be read and sent individually.
4. The length of each chunk of data that will be sent. Do not include the length of any trailing headers.

About this task

The procedure described in this topic enables you to create a correctly constructed chunked message, as defined in the HTTP/1.1 specification. See "The HTTP protocol" on page 12 for more information about the HTTP/1.1 specification. If the chunked message is not correctly constructed, the recipient may discard it.

"Sending an HTTP response from CICS as an HTTP server" on page 86 is the main set of instructions for writing an application program to send a server response. "Making HTTP requests through CICS as an HTTP client" on page 148 is the main set of instructions for writing an application program to make a client request. You can use the instructions in the present topic in conjunction with either of those sets of instructions.

The body of a chunked message cannot be formed directly from CICS documents (so the DOCTOKEN option cannot be used). The FROM option must be used to specify data to form the body of a chunked message.

When you have begun sending the parts of a chunked message, you cannot send any different messages or receive any items, until the final empty chunk is sent and the chunked message is complete.

Procedure

1. Before beginning a chunked message, verify that the Web client or server is at HTTP/1.1 version. All HTTP/1.1 applications are required to understand chunked transfer-coding. A chunked message cannot be sent to an HTTP/1.0 recipient.
   a. For responses sent by CICS as an HTTP server, use the WEB EXTRACT command to check the HTTP version specified for the Web client's request.
   b. For requests sent by CICS as an HTTP client, the HTTP version of the server is returned on the WEB OPEN command for the connection if you specify the HTTPVNUM and HTTPRNUM options on the command. If you did not do this, use the WEB EXTRACT command to check the HTTP version of the server.
   c. Alternatively, you can omit this check and allow CICS to check the version of the Web client or server when you issue the WEB SEND command to send the first chunk of the message. If the recipient is HTTP/1.0, CICS does not carry out the send, and returns an error response to you.
2. Use the WRITE HTTPHEADER command as many times as necessary to write any HTTP headers that should be sent *before* the body of the message. Do not write the headers for chunked transfer-coding; CICS writes these itself, using the chunk length information supplied by the application program.
3. If you want to include trailing headers (headers sent out *after* the body of the message) with the chunked message, use the WRITE HTTPHEADER command to write a Trailer header. Specify the names of all the HTTP headers you plan to send in the trailer, as the value of the Trailer header. You may send any headers as trailing headers, except the Transfer-Encoding, Trailer and Content-Length headers.
   a. For responses sent by CICS as an HTTP server, you need to ensure that the Web client sent a TE: trailers header on its request. This header shows that the client understands trailing headers. CICS returns an INVREQ response with a RESP2 value of 6 to the WRITE HTTPHEADER command if you attempt to write the Trailer header when the client did not send TE: trailers. Alternatively, you can use the READ HTTPHEADER command to check for the presence of the TE: trailers header.
   b. For requests sent by CICS as an HTTP client, trailing headers may be included without reference to the TE header.

   The trailing headers themselves are written during the chunked sending process.
4. Use the WEB SEND command to send the first chunk of the message.
   a. Specify CHUNKING(CHUNKYES) to tell CICS that this is a chunk of a message.
   b. Use the FROM option to specify the first chunk of data from the body of the message.
   c. Use the FROMLENGTH option to specify the length of the chunk.
   d. For requests by CICS as an HTTP client, an appropriate method must be specified on the METHOD option. Chunked transfer-coding is not relevant for requests with no message body, so it is not relevant for the GET, HEAD, DELETE, OPTIONS, and TRACE methods, but it can be used for the POST and PUT methods.
   e. Specify any other options that apply to both chunked and non-chunked messages, as given in your main set of instructions. For example, if this chunked message is the final message that you want to send to this server or Web client, specify the CLOSESTATUS(CLOSE) option.

5. Use the WEB SEND command as many times as necessary to send each of the remaining chunks of the message. On each WEB SEND command, just specify the following items:
   a. CHUNKING(CHUNKYES).
   b. The FROM option, giving the chunk of data.
   c. The FROMLENGTH option, giving the length of the chunk.

   Do not specify any of the other options for the command. CICS sends each chunk as you issue the command.
6. Optional: At any time after issuing the WEB SEND command for the first chunk, but before issuing the WEB SEND command for the final empty chunk (see the next step), use the WRITE HTTPHEADER command to create further HTTP headers that should be sent as trailing headers. Provided that a Trailer header was written on the first chunk of the message, the HTTP headers written during the chunked sending process are treated by CICS as trailing headers, and they are sent out with the final empty chunk. (If the Trailer header was not written, CICS does not allow any trailing headers to be written.) Note that CICS does not check whether your trailer headers match the names that you specified in the initial Trailer header on the first chunk of the message.
7. 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 generates and sends an empty chunk to the recipient to end the chunked message. The empty chunk is sent along with the trailer containing any trailing headers that you wrote.
8. For CICS as an HTTP server, errors are handled as follows:
   a. If one of the WEB SEND commands fails during the sequence, an error response is returned, and subsequent sends will also fail. The application should handle this situation appropriately.
   b. If all of the chunks are sent successfully but the application does not issue the final WEB SEND command with CHUNKING(CHUNKEND), the transaction is abended with abend code AWBP. This is necessary because CICS cannot guarantee that the chunked message is complete and correct, and so cannot issue the final empty chunk on behalf of the application.

   An incomplete chunked message should be ignored and discarded by the recipient. The Web client will decide whether or not to retry the request.
9. For CICS as an HTTP client, errors are handled as follows:
   a. If your application program is informed of an error at any point in the chunked transfer-coding process, use the WEB CLOSE command to stop the process and close the connection. The server will not receive the final empty chunk, and so should ignore and discard the data that you have sent so far. You can decide whether or not to retry the request.
   b. If you do not send the final empty chunk or issue the WEB CLOSE command, a warning message is written at task termination to CWBO, the transient data queue for CICS Web support messages. The server should time out the receive, and ignore and discard the data that you sent.

Managing application state across an HTTP request sequence

CICS initiates a new alias transaction and a new program for each request made by a Web client. This is the case for pipelined requests, requests made using a persistent connection, and requests that form a logical sequence, as well as for individual standalone requests. You need to consider how the application's state

will be managed between requests. If you need to share data across the request sequence, between different programs or instances of the same program, you can do this using CICS-managed resources, or using elements of the requests sent by the Web client.

About this task

When more than one exchange of a request and response between a Web client and CICS is needed to complete a task successfully, each new step in the sequence is initiated by the Web client. You can design the response sent by CICS to guide the Web client, and any human user of the Web client, to the next step. For example, the entity body can contain controls (such as links or buttons) that the end user can use to compose the next request. However, you cannot easily enforce the correct sequence of requests. In particular, the planned sequence can be disrupted if:

- The client is a Web browser, and the end user types a known URL to initiate a particular request, rather than selecting a control in an HTML page provided by a previous response.
- The end user abandons the activity altogether, by shutting down the Web client or by changing to some alternative activity with the Web client.

The end user might also delay initiation of any request in the sequence.

You should design your application programs so that they can cope with delays or disruptions in the request sequence. For example, if you are sharing data across the request sequence, you should ensure the data is cleaned up if the request sequence does not complete or is delayed excessively. If your application programs update protected resources, you should ensure that updates that must be committed or backed out together are made in the same transaction. (This means that a single request from the Web client should be designed to complete the update.)

The ideal situation for an application is that each exchange of a request and response is self-contained and completes an independent element of the task. However, this design is not always possible, especially when the task is complex, or when a Web client has sent a pipelined sequence of requests. A pseudoconversational model may be required, where the application's state must be managed between requests. This can be arranged using the following techniques:

Procedure

- You can design the requests sent by the Web client so that application state, or shared data, is incorporated in the request, for example as part of a request URL that is used when the Web client submits an HTML form. The next program can examine the request URL to obtain the shared data.
- You can store small quantities of application state using hidden fields in an HTML form that is returned to the Web client as a response. When the user performs the next action in the planned sequence, the request that they send to CICS can include the hidden fields, which can be located and read by the next application program.
- For larger quantities of state, and state with an extended lifetime, you can create a CICS-managed resource to maintain the application's state, and pass a token that represents the resource. CICS provides sample state management programs (DFH$WBST and DFH$WBSR) that store application state in main storage or temporary storage queues, and provide tokens that application programs can use to access the information. A token can be conveyed from program to

program in a pseudoconversation as a hidden field in an HTML form, or from interaction to interaction as a query string in a URL. This technique can be used for preserving information throughout a pseudoconversation, and also for preserving information throughout an extended interaction between an end user and various CICS application programs, perhaps over several pseudoconversations.

Chapter 7. Resource definition for CICS as an HTTP server

About this task

The CICS-supplied resource definition group DFHWEB contains the following CICS Web support resources:

- Transaction definitions for CICS Web support tasks (for example, CWBA and CWXN)
- CICS Web support utility programs, including:
  - The default analyzer program DFHWBAAX and the sample analyzer program DFHWBADX
  - The Web error programs DFHWBEP and DFHWBERX
- A temporary storage model, DFHWEB

The transient data queues for CICS Web support messages, CWBO (for most messages) and CWBW (a separate queue for warning header messages), are in the group DFHDCTG.

The resource definition group DFH$WEB contains most of the PROGRAM resource definitions and URIMAP definitions for the sample CICS Web support applications.

You need to create some additional resource definitions for each CICS Web support task that you want to perform. Chapter 5, "Planning your CICS Web support architecture for CICS as an HTTP server," on page 59 has planning guidance that specifies the resource definitions which you need for each task. The resource definitions that you might need to set up are as follows:

Procedure

- Create a TCPIPSERVICE resource definition to define each port that you use to receive inbound HTTP requests for CICS Web support. This resource definition is the place where you specify the security measures that are applied for each port, along with technical information on the operation of the port. "Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 tells you how to do this.
- Optional: Create TRANSACTION resource definitions for any alias transactions that you want to use for inbound HTTP request processing. "Creating TRANSACTION resource definitions for CICS Web support" on page 99 tells you how to do this.
- Create a URIMAP resource definition to provide processing information for each HTTP request for CICS as an HTTP server.
  1. For all HTTP requests to CICS as an HTTP server, start the definition following the steps in "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100.
  2. For HTTP requests where an application-generated response is provided, follow the steps in "Completing a URIMAP definition for an application response to an HTTP request for CICS as an HTTP server" on page 102.
  3. For HTTP requests where a static response is provided, follow the steps in "Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server" on page 103.

Creating TCPIPSERVICE resource definitions for CICS Web support

Use TCPIPSERVICE resource definitions to define the association between ports and CICS services, including CICS Web support. Define and install a TCPIPSERVICE resource definition for each port that you use for CICS Web support.

About this task

Each TCPIPSERVICE definition that is active in a CICS system must specify a unique port number. CICS uses the TCPIPSERVICE definition for a port to determine which CICS service is invoked when it receives an inbound TCP/IP connection request on that port. Use the PROTOCOL attribute to identify the service. Specify HTTP for standard CICS Web support, and USER for non-HTTP requests that are handled using CICS Web support.

For CICS Web support, create TCPIPSERVICE definitions for the default, or well-known, port numbers that are used for Internet services. For HTTP, the default port number is 80, and, for HTTPS, the default port number is 443. You can also use nonstandard port numbers.

Each TCPIPSERVICE definition can specify only one analyzer program and one transaction definition for the Web attach task. If you need to use more than one of these items, you must use different TCPIPSERVICE definitions and, therefore, different ports.

CICS provides sample TCPIPSERVICE definitions for CICS Web support in group DFH$SOT:

HTTPNSSL
: CICS Web TCPIPSERVICE with no SSL support

HTTPSSL
: CICS Web TCPIPSERVICE with SSL support

Important: Use the TCPIPSERVICE resource definition to specify the security measures that are applied for each port. You can choose whether or not to use SSL, and, if you do use SSL, you choose the exact security measures that are applied; for example, the authentication method, the sending of certificates by client and server, and the encryption of messages. See Chapter 13, “Security for CICS Web support,” on page 173 for more information about the security features that you can use to keep your CICS Web support facility safe.

The *CICS Resource Definition Guide* describes the different methods of resource definition, and provides reference information about all the TCPIPSERVICE resource definition attributes that you will use during this process.

Procedure

1. Identify a TCP/IP port to use for CICS Web support. You are recommended to reserve the port number for use by CICS Web support. See “Reserving ports for CICS Web support” on page 54 for information on port usage.
2. Begin a TCPIPSERVICE definition with a name and group of your choice, using one of the methods listed in the *CICS Resource Definition Guide*. When you set up URIMAP definitions for inbound HTTP requests on this port, specify the name of the TCPIPSERVICE definition.

3. Use the STATUS attribute to specify whether CICS starts listening for this service immediately after the definition is installed. If you specify CLOSED, you must set the service open before it can be used. You can set the service open or closed using the CEMT transaction or the **SET** TCPIPSERVICE system programming command.
4. Specify the PORTNUMBER attribute as the number of the TCP/IP port that is covered by this definition.
5. Use the HOST attribute to specify the dotted decimal or colon hexadecimal IP address on which the TCPIPSERVICE listens for incoming connections. You can also use the IPADDRESS attribute to specify the dotted decimal IP address for existing programs. Alternatively, for configurations with more than one IP stack, you can specify INADDR_ANY to make CICS try to bind to the port on every stack where it is defined. Or, if you have a multi-stack CINET environment, and you want to assign affinity only to the default TCP/IP stack, you can specify DEFAULT to do this. The reference information about this TCPIPSERVICE resource definition attribute details some additional considerations, which are important if you want more than one CICS region to share this TCPIPSERVICE definition, or if you want more than one CICS region to bind to the port number that it specifies.
6. Use the DNSGROUP and GRPCRITICAL attributes to specify the DNS group name that the service uses in the sysplex domain, and the critical status for the service. This information enables CICS to register to Workload Manager for DNS connection optimization. *Java Applications in CICS* has more information about this area.
7. Use the PROTOCOL attribute to specify that CICS Web support handles requests on this port.
   a. Specify HTTP for normal HTTP requests. CICS forces HTTP if you specify ports 80 or 443. This option covers both HTTP with SSL and HTTP without SSL. The SSL option specifies whether SSL is involved.
   b. Specify USER for non-HTTP requests that are handled using CICS Web support. When you specify USER, CICS Web support is used for handling the request, but no acceptance checks are carried out for messages sent and received using this protocol. The requests are flagged as non-HTTP and passed straight to the analyzer program. URIMAP definitions are not used for these requests.
8. Specify the TRANSACTION attribute as the 4-character ID of the Web attach task, which is normally CWXN for HTTP requests or CWXU for non-HTTP (USER protocol) requests. This task handles initial processing of a request. CICS provides CWXN as a default if you specify ports 80 or 443. If required for accounting or monitoring purposes, you can specify an alias of CWXN or CWXU, both of which must run the program DFHWBXN.
9. Specify the URM attribute as the name of the analyzer program that is associated with this TCPIPSERVICE definition. For a non-HTTP (USER protocol) request, the analyzer program is always used. For an HTTP request, the analyzer program is used to interpret the request if a URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present. You must specify an analyzer program. You can select only one analyzer program for each TCPIPSERVICE definition, but you can code it to handle any requests. Chapter 10, “Analyzer programs,” on page 125 tells you about the basic support that your analyzer program must provide if you intend to use URIMAP definitions to handle all your HTTP requests. The architecture guidance in Chapter 5, “Planning your CICS Web support architecture for CICS as an HTTP server,” on page 59 helps you to decide whether to involve the analyzer program for any particular HTTP request.

10. Use the SOCKETCLOSE attribute to specify how long CICS waits before closing the socket after issuing a receive for incoming data on that socket. NO means that the socket is left open until data is received or until the Web client closes it. To prevent the socket from being blocked by a slow or broken Web client, specify a timeout value rather than specifying NO. On the first receive command issued by the Web attach task after a connection is made, this timeout value is ignored, and the task waits to receive data from the Web client for a period of time determined by CICS (30 seconds for HTTP). This delay prevents a socket connection from being closed as soon as it is started, even if no data is immediately available, and so prevents a connection reset error at the Web client.

    Note: For CICS Web support, never specify a zero setting for SOCKETCLOSE. SOCKETCLOSE(0) means that a persistent connection cannot be maintained, even if the Web client requests it.
11. Use the BACKLOG attribute to specify the number of connections that can be queued before TCP/IP starts to reject incoming requests from Web clients. The default is 1.
12. Use the MAXDATALEN attribute to specify the maximum length of data that can be received on this connection. The default value is 32 KB and the maximum is 524 288 KB. This option helps to guard against denial of service attacks involving the transmission of large amounts of data.
13. Use the SSL attribute to specify whether the secure sockets layer (SSL) is used for this port. YES means that SSL is used, and CICS sends a server certificate to the Web client. CLIENTAUTH means that SSL is used, and that the Web client is requested to send a client certificate to CICS, in addition to CICS sending a server certificate to the Web client. CICS provides YES as a default if you specify port number 443, and forces NO if you specify port number 80. Chapter 13, “Security for CICS Web support,” on page 173 explains what to do if you are using SSL.
14. If you have specified SSL(YES) or SSL(CLIENTAUTH), use the CERTIFICATE attribute to specify the label of an X.509 certificate that CICS uses as the server certificate during the SSL handshake. 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 database. Chapter 13, “Security for CICS Web support,” on page 173 has more information about using certificates.
15. Use the AUTHENTICATE attribute to specify the level of authentication that is used for Web clients making requests on this port. Chapter 13, “Security for CICS Web support,” on page 173 explains authentication and identification.
    a. Specify NO if the Web client is not required to send authentication or identification information. If the client sends a valid certificate that is already registered to the security manager, CICS can use it.
    b. Specify BASIC to make CICS attempt HTTP basic authentication, where CICS requests a user ID and password from the Web client. “HTTP basic authentication” on page 20 explains basic authentication in more detail.
    c. Specify CERTIFICATE to use SSL client certificate authentication. The Web client must send a valid certificate that is already registered to the security manager and associated with a user ID. If a valid certificate is not received, or the certificate is not associated with a user ID, the connection is rejected. You must specify SSL(CLIENTAUTH) if you use this option.
    d. Specify AUTOREGISTER to use SSL client certificate authentication with auto-registration for the security manager. The Web client must send a valid certificate. If CICS finds that the certificate is not yet registered to the

security manager, HTTP basic authentication is used to request a user ID and password, and CICS uses this information to register the certificate. You must specify SSL(CLIENTAUTH) if you use this option.

e. Specify AUTOMATIC to use SSL client certificate authentication with auto-registration for the security manager (as for the AUTOREGISTER option), or, if no certificate is sent, to use HTTP basic authentication (as for the BASIC option).

16. Use the REALM attribute to specify the realm that is used for HTTP basic authentication. The end user sees the realm during the process of basic authentication. It identifies the set of resources to which the authentication information requested (that is, the user ID and password) apply.

    a. If you require different authentication information for resources delivered using different TCPIPSERVICE definitions, specify different realms to make this requirement clear to the end user.

    b. If end users use the same authentication information across your resources, you can specify the same realm on multiple TCPIPSERVICE definitions.

    c. If you do not specify the REALM attribute, the default realm is used. The default realm is:

    ```
    realm="CICS application aaaaaaaa"
    ```

    where *aaaaaaaa* is the applid of the CICS region.

Creating TRANSACTION resource definitions for CICS Web support

TRANSACTION resource definitions are used to define alias transactions for CICS Web support. An alias transaction handles the later stages of processing for an HTTP request, including receiving the request, executing the application's business logic, construction of the HTTP response and code page conversion of the HTTP response. Alias transactions can also be used for processing non-HTTP requests.

About this task

CICS supplies a resource definition for a default alias transaction, CWBA. You may want to use alternative alias transaction names for the purposes of:

- Auditing, monitoring or accounting
- Resource and command checking for security
- Allocating initiation priorities
- Allocating DB2® resources
- Assigning different runaway values to different CICS application programs
- Transaction class limitation

You can set up as many alias transaction definitions as you want. You can use the URIMAP definition or an analyzer program to specify the alias transaction that is required for a particular request.

Important: 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). The *CICS Performance Guide* explains why this is important.

The *CICS Resource Definition Guide* has full instructions for this type of resource definition. When you are following these instructions, remember:

Procedure

- Base your alias transaction definition on the definition of CWBA, making any changes that you require, such as changes to priority. The definition of CWBA is:

```
DEFINE TRANSACTION(CWBA)    GROUP(DFHWEB)
       PROGRAM(DFHWBA)      TWASIZE(0)
       PROFILE(DFHCICST)    STATUS(ENABLED)
       TASKDATALOC(BELOW)   TASKDATAKEY(USER)
       RUNAWAY(SYSTEM)      SHUTDOWN(ENABLED)
       PRIORITY(1)          TRANCLASS(DFHTCL00)
       DTIMOUT(NO)          INDOUBT(BACKOUT)
       SPURGE(YES)          TPURGE(NO)
       RESSEC(NO)           CMDSEC(NO)
```

- Your alias transaction definition must use the CICS-supplied alias program DFHWBA . The alias program calls the user application program that you have specified to process the request.
- Your alias transaction definition must be a local transaction.

Starting a URIMAP resource definition for any requests for CICS as an HTTP server

URIMAP resource definitions are used to define how HTTP requests are processed. For any HTTP request for CICS as an HTTP server, start the URIMAP definition by specifying the components of the expected URL for the Web client's request (scheme, host and path) and other basic information.

Before you begin

If you have not already planned how to provide a response to the HTTP request for CICS as an HTTP server, “Providing dynamic HTTP responses with Web-aware application programs” on page 59 and “Providing static HTTP responses with a CICS document template or z/OS UNIX file” on page 64 tell you how to do this.

About this task

The *CICS Resource Definition Guide* has information about the different methods of resource definition and full reference information about all the URIMAP resource definition attributes that you use during this process.

Procedure

1. Identify the URL that you plan to receive as an HTTP request from a Web client. The URL represents a resource that you plan to make available to a Web client through CICS.
2. Divide the URL for the HTTP request into its scheme, host and path components. “The components of a URL” on page 10 explains each of these components and how they are delimited. For example, in the URL `http://www.example.com/software/index.html`:
   - The scheme component is `http`
   - The host component is `www.example.com`
   - The path component is `/software/index.html`

   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/*` would make the URIMAP definition match

all requests whose path begins with the string `/software/`. If more than one wildcarded URIMAP definition matches an HTTP request, the most specific match is taken.

3. If a query component is present in the URL, and you want to match the URIMAP definition to that specific query alone, you can include this as part of the PATH specification. A query string can be used after a path that includes an asterisk as a wildcard, but the query string cannot itself include an asterisk as a wildcard. The complete and exact query string must be specified. For a static response with a CICS document template, a query string can either be used to select the URIMAP definition, or it can be substituted into the document template, but not both. If you do not include a query string in the URIMAP definition, matching takes place only on the path, and any query string that is present in the request is automatically ignored for matching purposes.
4. Begin a URIMAP definition with a name and group of your choice, as specified in the *CICS Resource Definition Guide*.
5. Use the STATUS attribute to specify whether the URIMAP definition is installed in an enabled or disabled state.
6. Specify a USAGE attribute of SERVER (CICS as an HTTP server).
7. Specify the SCHEME attribute as the scheme component of the URL for the HTTP request. You can use HTTP (without SSL) or HTTPS (with SSL). Do not include the delimiters `://` following the scheme component. A URIMAP specifying the HTTP scheme accepts Web client requests made using either the HTTP scheme or the more secure HTTPS scheme. A URIMAP specifying the HTTPS scheme accepts only Web client requests made using the HTTPS scheme.
8. Optional: Specify the TCPIPSERVICE attribute as the name of the TCPIPSERVICE definition that defines the inbound port to which this URIMAP definition relates. If you do not specify this attribute, the URIMAP definition applies to a matching HTTP request on any inbound port. When a URIMAP definition with HTTPS (HTTP with SSL) 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.
9. If you need to distinguish between URLs containing different host names, specify the HOST attribute as the host component of the URL for the HTTP request. Do not include a port number. An IPv4 or IPv6 address can be used as a host name. If you specify a single asterisk as the HOST attribute, the URIMAP definition matches any host name on incoming URLs. Use this option if you are not using multiple host names or if you do not want to distinguish them.
10. Specify the PATH attribute as the path component of the URL for the HTTP request, including an asterisk as a wildcard character if required. You can either include or omit the delimiter / (forward slash) at the beginning of the path component; if you omit it, CICS automatically provides it. If a query component is present, and you want to apply the URIMAP definition to that specific query alone, include this as part of the path component (including the question mark at the beginning of the string).

Note: Specifying the PATH attribute as /* makes the URIMAP definition match all requests directed to the host named in the HOST attribute, unless more specific URIMAP definitions are also installed, in which case the most specific match is taken.

11. Now complete your definition:
    a. For an application-generated response, follow the instructions in "Completing a URIMAP definition for an application response to an HTTP request for CICS as an HTTP server."
    b. For a static response, follow the instructions in "Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server" on page 103.

Completing a URIMAP definition for an application response to an HTTP request for CICS as an HTTP server

If you are providing an application-generated response to an HTTP request, when you have started the URIMAP definition by specifying the components of the expected URL (scheme, host and path) and other basic information, complete the definition by providing information about the application or applications that should process the request and supply an HTTP response.

Before you begin

If you have not already planned how to provide a response to the HTTP request for CICS as an HTTP server, "Providing dynamic HTTP responses with Web-aware application programs" on page 59 tells you how to do this. Then you need to start your URIMAP definition as described in "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100.

About this task

When you have planned your application-generated response and started your URIMAP definition, complete the definition following the instructions in this topic. the *CICS Resource Definition Guide* has information about the different methods of resource definition, and full reference information about all the URIMAP resource definition attributes that you will use during this process.

Procedure

1. If the analyzer program associated with the TCPIPSERVICE definition (or definitions) to which this URIMAP definition relates is to be involved in the processing path for this request, specify YES for the ANALYZER attribute to activate it. If an analyzer program is used, you can still specify the CONVERTER, TRANSACTION, USERID and PROGRAM attributes. The values that you specify for these attributes are used as input to the analyzer program, but they can be overridden by it. Alternatively, you can leave these attributes blank and let the analyzer program specify them.
2. If you are using a converter program, specify the CONVERTER attribute as the name of the program. The program can be any converter program that is available in CICS; there is no association between the converter program and the TCPIPSERVICE definition, as there is for the analyzer program. If a converter program is used, you can still specify the PROGRAM attribute. The value that you specify for this attribute is used as input to the converter program. The converter program can change the PROGRAM attribute to specify a different application program to process the request.

3. Specify the TRANSACTION attribute as the name of an alias transaction that is available in CICS, which CICS can use to run the application program that provides the response. The default alias transaction is CWBA. The transaction name can also be changed or provided by an analyzer program, if ANALYZER(YES) is specified.
4. Specify the USERID attribute as a default user ID under which the alias transaction can be attached. When authentication is required for the connection, so that CICS requests an authenticated user ID directly from the client, the default user ID is not used. The authenticated user ID of the client is used instead, or if authentication fails, the request is rejected. If you use an analyzer program, it can replace a default user ID or an authenticated user ID with another user ID, or provide one. If no user ID is specified, the default user ID is the CICS default user.
5. Specify the PROGRAM attribute as the name of the application program that provides the response to the request. If no analyzer or converter program is specified in the URIMAP definition, the HTTP request is passed directly to this application program. If an analyzer or converter program is specified, you can leave this attribute blank and let the analyzer or converter program specify it.

Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server

For a static response, when you have started the URIMAP definition by specifying the components of the expected URL (scheme, host and path) and other basic information, complete the definition by providing the information that CICS needs to construct the static response to the request, using a document template or z/OS UNIX file. You can use a wildcard for path matching, where 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 or z/OS UNIX file path.

Before you begin

If you have not already planned how to provide a static response to the HTTP request, "Providing static HTTP responses with a CICS document template or z/OS UNIX file" on page 64 tells you how to do this. Then you need to start your URIMAP definition as described in "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100.

About this task

When you have planned your static response and started your URIMAP definition, complete the definition following the instructions in this topic. The *CICS Resource Definition Guide* has information about the different methods of resource definition, and full reference information about all the URIMAP resource definition attributes that you will use during this process.

Note: The URIMAP definition is not used to control security for CICS document templates and z/OS UNIX files delivered as static responses. If you want to use basic authentication and resource level security to secure these items, Chapter 13, "Security for CICS Web support," on page 173 explains how to set these up.

Procedure

1. Specify the MEDIATYPE attribute as the data content of the static response that CICS provides. For example, `text/html` or `text/xml` are the names for HTML and XML data content respectively. (See "IANA media types and character

sets" on page 10 for more information about media types.) There is no default for this attribute, and it must be specified. CICS creates a Content-Type header for the response using this information.

2. If the static response is formed from a text document (either a document template or a z/OS UNIX file), specify the attributes required for code page conversion. Code page conversion only takes place where the MEDIATYPE attribute specifies a text type of data content.
   a. Specify the CHARACTERSET attribute as the character set into which CICS converts the static response before sending it to the Web client. CICS does not support all the character sets named by IANA. Appendix A, "HTML coded character sets," on page 373 lists the IANA character sets that are supported by CICS. This information is included in the Content-Type header of the response.
   b. Specify the HOSTCODEPAGE attribute as the IBM code page (EBCDIC) in which the static document is encoded.
3. If you are using a CICS document template to form the static response, specify the TEMPLATENAME attribute as the name of the document template. The document template must be defined using a DOCTEMPLATE resource definition. 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. The *CICS Resource Definition Guide*URIMAP definition attributes has an example of how this works.

   When the TEMPLATENAME attribute is specified, if a query string is present on the URL, CICS passes the content of the query string into the named CICS document template as a symbol list. This only takes place if the query string has not already been used in the PATH attribute of the URIMAP definition.
4. If you are using a z/OS UNIX file to form the static response, specify the HFSFILE attribute as the fully qualified (absolute) or relative name of the file. The z/OS UNIX file can be specified as an absolute, or fully qualified, path that begins with a slash, or as a relative path that does not begin with a slash. A relative path is relative to the HOME directory of the CICS region userid. The CICS region must have permissions to access z/OS UNIX, and it must have permission to access the z/OS UNIX directory containing the file, and the file itself. *Java Applications in CICS* explains how to grant these permissions. If you want to use path matching, include an asterisk as a wildcard character at the end of the path for the z/OS UNIX 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 z/OS UNIX file path. The *CICS Resource Definition Guide* has an example of how this works.

   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 a z/OS UNIX file.

Chapter 8. Administering CICS as an HTTP server

When you have configured CICS to perform a variety of CICS Web support tasks, and started to respond to requests from Web clients, you might need to carry out some administrative activities to manage your CICS Web support structure and to provide appropriate handling for requests if a resource is unavailable.

About this task

Administration for CICS as an HTTP server is facilitated by having URIMAP definitions to manage your HTTP requests. URIMAP definitions enable you to:

- Redirect or reject specific HTTP requests dynamically in a running CICS system, if the resources needed by those requests (for example, a CICS program) are not available.
- Have virtual hosts created by CICS, which can be managed using CICS commands.

If you do not have URIMAP definitions, you can administer CICS Web support at the level of a TCPIPSERVICE resource definition, which manages all requests on a particular port, but managing at the level of the URIMAP resource definition gives greater control and granularity.

Procedure

- You can use the CICS system programming interface and CICS-supplied transactions to create, install, update and delete CICS Web support resources, including TCPIPSERVICE, URIMAP, and TRANSACTION resource definitions. “Managing CICS Web support resources” on page 106 explains the commands that you can use to administer these resources.
- You can use the INQUIRE HOST command and the virtual host browsing commands to see the virtual hosts that CICS creates from your URIMAP definitions, and the **SET** HOST command to change their status. “Administering virtual hosting” on page 107 tells you how to do this.
- If an application or resource in your CICS system is temporarily unavailable and you want to provide cover through redirection, you can redirect HTTP requests that are handled by a URIMAP definition, and remove the redirection when the resource becomes available again. If the resource's location or request URL format has changed permanently, you can set up a permanent redirection. “Redirecting HTTP requests to another URL” on page 108 tells you how to do this.
- If an application or resource in your CICS system is temporarily unavailable and you cannot provide cover through redirection, or if an application or resource has been permanently removed, you can reject HTTP requests at several different levels (individual request URL, virtual host, port, or all ports) by disabling resource definitions. You can use these methods to terminate all or part of your CICS Web support service. “Rejecting HTTP requests” on page 109 tells you how to do this.
- You might want to provide a favicon or a robots.txt file for each of your hosts. These items are requested automatically by many Web browsers. “Providing a favorites icon” on page 110 and “Providing a robots.txt file” on page 112 tell you how to provide these.

- CICS records information from Warning headers in inbound messages to a transient data queue, CWBW. The information is normally meant to be read by a user. “Warning headers” on page 114 tells you about this information.

Managing CICS Web support resources

You can use the CICS system programming interface and the CICS-supplied transactions CEMT and CEDA to create, install, update and delete CICS Web support resources.

About this task

The CICS-supplied transaction CEDA can be used to create and install TCPIPSERVICE, URIMAP, TRANSACTION and DOCTEMPLATE resource definitions. Chapter 7, “Resource definition for CICS as an HTTP server,” on page 95 has more information about setting up resource definitions for CICS Web support.

The CICS system programming interface includes the following commands for CICS Web support administration:

INQUIRE TCPIP
Inquire on the status of TCP/IP support in the CICS system. The command returns the open status for TCP/IP and the maximum and current number of IP sockets in the CICS region.

SET TCPIP
You can use this command to open or close TCP/IP support, either normally, with active tasks being allowed to complete, or immediately, with active tasks being terminated abnormally. Closing TCP/IP support means that all inbound and outbound requests are rejected, and CICS Web support is stopped completely. You can also use this command to increase or reduce the maximum number of IP sockets in the CICS region. If you do not have superuser authority, the limit that you can set is lower, and CICS informs you if it has imposed this lower limit.

CREATE TCPIPSERVICE
Create a TCPIPSERVICE definition for a port.

DISCARD TCPIPSERVICE
Delete the TCPIPSERVICE definition for a port. The TCPIPSERVICE definition must be closed (using the **SET** TCPIPSERVICE command) before it can be discarded.

INQUIRE TCPIPSERVICE
Inquire on a TCPIPSERVICE definition. As well as displaying the attributes of the definition, the inquiry shows the current DNS registration status and open status for the definition. TCPIPSERVICE definitions may also be browsed.

SET TCPIPSERVICE
You can use this command to change the request queue limit, the data receive limit, the DNS registration status, or the analyzer program for the TCPIPSERVICE definition. You can also use this command to close the TCPIPSERVICE definition. You can choose to stop listening on the port normally, with active tasks being allowed to complete, or immediately, with active tasks being terminated abnormally.

CREATE URIMAP
Create a URIMAP definition for a request.

DISCARD URIMAP
Delete a URIMAP definition. Requests that were handled by the deleted definition might be matched by a less specific URIMAP definition that has a wildcard character in the path. Otherwise, they will pass to the analyzer program for the TCPIPSERVICE definition. If you want to reject the requests without this alternative handling, disable the URIMAP definition rather than discarding it.

INQUIRE URIMAP
Inquire on a URIMAP definition. As well as displaying the attributes of the definition, the inquiry shows if a URIMAP has been disabled on an individual basis, or if it is unavailable because the virtual host of which it is a part has been disabled. URIMAP definitions may also be browsed.

SET URIMAP
You can use this command to enable or disable a URIMAP definition. When a URIMAP definition is disabled, CICS returns an HTTP 503 response (Service Unavailable) to the Web client through a Web error program. You can also use this command to set the LOCATION and REDIRECTTYPE attributes to specify redirection, or to end a redirection.

INQUIRE HOST
Inquire on a virtual host. Virtual hosts may also be browsed. "Administering virtual hosting" explains how to manage virtual hosts.

SET HOST
Enable or disable a virtual host. "Administering virtual hosting" explains how to manage virtual hosts.

TRANSACTION, DOCTEMPLATE and PROGRAM resources that you use for CICS Web support can also be managed using SPI commands. *CICS System Programming Reference* has full information about all these commands.

The CICS-supplied transaction CEMT includes the following commands for CICS Web support administration:

- INQUIRE TCPIP
- **SET** TCPIP
- INQUIRE TCPIPSERVICE
- **SET** TCPIPSERVICE
- INQUIRE URIMAP
- **SET** URIMAP
- INQUIRE HOST
- **SET** HOST

TRANSACTION and PROGRAM resources that you use for CICS Web support can also be managed using CEMT, and you can inquire on DOCTEMPLATE resources. *CICS Supplied Transactions* has full information about all these commands, and explains how to use CEMT.

CICSPlex® SM can also be used to manage the resources listed here.

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.

Redirecting HTTP requests to another URL

You can redirect an HTTP request for CICS as an HTTP server to another URL using a URIMAP definition.

About this task

You might intend that the resource should always be provided by redirecting the Web client to another URL. Alternatively, you might want to use redirection to provide a temporary response to a request while the intended resource is unavailable (for example, a page telling the requester that the application they need is offline). In either case, you can redirect the request using a URIMAP definition that matches the request, as follows:

Procedure

1. Locate the URIMAP definition for the URL that you want to redirect.
2. Use the LOCATION attribute of the URIMAP definition to specify a URL of up to 255 characters, to which matching HTTP requests are redirected. This must be a complete URL, including scheme, host and path components. Include all the delimiters. CICS checks that the URL is complete and correctly delimited, but CICS does not check that the destination is valid.
   a. Optional: You can use a fragment identifier (preceded by a # character) in the LOCATION attribute, to point a Web browser to a reference or function within the item identified by the URL. For example, a fragment identifier can be the ID of a subsection within a document. Consult the technical specification for the type of content that you are providing (for example, HTML) to see whether and how fragment identifiers can be used.
3. Use the REDIRECTTYPE attribute of the URIMAP definition to specify temporary or permanent redirection. When requests are redirected on a temporary basis, the HTTP status code used for the response is 302 (Found). When requests are redirected permanently, the HTTP status code used for the response is 301 (Moved Permanently). CICS composes the redirection response, and it cannot be customized.
4. Install the changed URIMAP definition. When REDIRECTTYPE(TEMPORARY) or REDIRECTTYPE(PERMANENT) is specified, the LOCATION attribute of the URIMAP definition overrides any other attributes in the URIMAP definition, and redirects the HTTP requests. You can use the **SET** URIMAP LOCATION command to change the LOCATION attribute after the URIMAP definition is installed.
5. If and when the resource becomes available again, use the command **SET** URIMAP REDIRECTTYPE(NONE) to switch off redirection, and re-install the changed definition. The URL specified in the LOCATION attribute is retained, but is not used unless you reactivate redirection.

Rejecting HTTP requests

Sometimes you might need to reject requests that Web clients make to CICS as an HTTP server, if an application or resource in the CICS system is unavailable.

About this task

You can reject HTTP requests at several different levels:

1. At the level of the specific request URL. To achieve this level of granularity, the request URL should be covered by a URIMAP definition. If you do not have URIMAP definitions, you can modify the handling of HTTP requests through changes to the analyzer program that handles the requests, but this is less convenient.
2. At the level of a virtual host (which covers all requests for a particular host name). For a request to be incorporated into a virtual host, it must be covered by a URIMAP definition.
3. At the level of a port. A port maps to a TCPIPSERVICE definition. For example, disabling the TCPIPSERVICE definition for the default HTTP port 80 would prevent CICS from receiving any HTTP requests, except those that use SSL or those that use non-standard ports.
4. Completely, at the level of all ports. In the CEMT transaction or in CPSM, you can shut down CICS internal TCP/IP sockets support, and so shut down CICS Web support completely.

Generally, if you reject the HTTP request at a more granular level, CICS can give a more appropriate and informative error response to the Web client. For example, if you reject an HTTP request by disabling a URIMAP definition or a virtual host, CICS returns an HTTP 503 response (Service Unavailable) to the Web client through a Web error program. You can tailor the Web error program to modify this response. However, if you reject HTTP requests by disabling a TCPIPSERVICE definition, the Web client will only receive a general error response that indicates a server error.

Procedure

- To reject requests to a particular request URL:
  1. If you have a URIMAP definition for the URL, disable the URIMAP definition using one of the methods described in "Managing CICS Web support resources" on page 106. Check that the request URL will not be matched by a less specific URIMAP definition that has a wildcard character in the path. CICS returns an HTTP 503 response (Service Unavailable) to the Web client through a Web error program. You can tailor this response by changing the Web error program.
  2. If you do not have a URIMAP definition for the URL, you can reject requests by changing the analyzer program associated with the TCPIPSERVICE definition for the port on which the request is made. You might want to code the analyzer program to provide an individual rejection message for each URL, or you might prefer to provide a single message that covers any URL that is unavailable. Chapter 10, "Analyzer programs," on page 125 tells you what actions are appropriate for handling rejected requests.
- To reject requests to a virtual host (that is, all requests to a certain host name), disable the virtual host using the **SET** HOST command, as described in "Administering virtual hosting" on page 107. CICS returns an HTTP 503 response (Service Unavailable) to the Web client through a Web error program.You can tailor this response by changing the Web error program.
- To reject all requests on a particular port, disable the TCPIPSERVICE definition using one of the methods described in "Managing CICS Web support resources" on page 106. You can choose to stop listening on the port normally, with active tasks being allowed to complete, or immediately, with active tasks being terminated abnormally.
- To reject all inbound and outbound requests and stop CICS Web support completely, use the CEMT transaction or CPSM to close TCP/IP, as described in "Managing CICS Web support resources" on page 106. You can choose to close normally, with active tasks being allowed to complete, or immediately, with active tasks being terminated abnormally.

Providing a favorites icon

Many Web browsers automatically make a request for a favorites icon (favicon) when a user visits or bookmarks a Web page. You can provide a favicon as a static response using a URIMAP definition.

About this task

A favicon, also known as a website icon, shortcut icon, url icon, or bookmark icon is a 16×16 pixel, 32×32 pixel or 64×64 pixel square icon associated with a particular website or webpage. A web designer can create a favicon and install it into a website or webpage, and most graphical web browsers will then use the favicon.

Web browsers make requests for default favicons using the URL
`http://www.example.com/favicon.ico`

where `www.example.com` is the host name for the site. The HTTPS scheme may be used instead, if appropriate. You can choose to provide:

- A default favicon that is returned for any host name used by your CICS region.
- A different default favicon for each host name used by your CICS region.

If a Web browser requests a favicon and you do not provide one, CICS sends an error response to the browser as follows:

- If you are using the CICS-supplied default analyzer DFHWBAAX, a 404 (Not Found) response is returned. No CICS message is issued in this situation.
- If you are using the sample analyzer DFHWBADX, or a similar analyzer which is only able to interpret the URL format that was required before CICS TS Version 3, the analyzer is likely to misinterpret the path `favicon.ico` as an incorrectly specified converter program name. In this case, message DFHWB0723 is issued, and a 400® (Bad Request) response is returned to the browser. To avoid this situation, you can either modify the analyzer program to recognize the favicon request and provide a more suitable error response, or provide a favicon using a URIMAP definition (which means that the sample analyzer program is bypassed for these requests).

To provide a favicon for all or some of your host names, using a URIMAP definition:

Procedure

1. Create the favicon and store it in a suitable location on z/OS UNIX file system.
   a. You can create the favicon using an icon editor package, or use an icon converter program to convert an image created in another format.
   b. The favicon must be 16 by 16 pixels. Browsers may ignore favicons that are not the correct size.
   c. The favicon must be saved in Windows icon format (.ico file extension), and named favicon.ico.

   Most Web servers need to store the favicon in the root directory for the host name. For CICS, a URIMAP definition can provide a favicon stored anywhere on z/OS UNIX. The CICS region must have permissions to access z/OS UNIX, and it must have permission to access the z/OS UNIX directory containing the file, and the file itself. *Java Applications in CICS* explains how to grant these permissions.
2. Create a URIMAP definition to provide the favicon as a static response. "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100 and "Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server" on page 103 guide you through the process of creating a URIMAP definition. The following sample URIMAP definition attributes could be specified to provide a favicon for all host names used by the CICS region:

```
 Urimap       ==> favicon        - URIMAP name
 Group        ==> MYGROUP        - Any suitable
 Description  ==> Favicon
 STatus       ==> Enabled
 USAge        ==> Server         - For CICS as HTTP server
UNIVERSAL RESOURCE IDENTIFIER
 SCheme       ==> HTTP           - Will also match HTTPS requests
 HOST         ==> *              - * matches any host name.
```

```
                                        Specify host name if you
                                        provide different favicons
 PAth         ==> /favicon.ico   - Browsers use this path to
                                        request favicons
ASSOCIATED CICS RESOURCES
 TCpipservice ==>                - Blank matches any port
STATIC DOCUMENT PROPERTIES
 Mediatype    ==> image/x-icon   - This media type is suitable
 HFsfile      ==> /u/cts/CICSHome/favicon.ico
                                        - Location of favicon in HFS
```

Note: Code page conversion is not required for a favicon, so do not specify the CHARACTERSET or HOSTCODEPAGE options.

Providing a robots.txt file

Web robots are programs that make automatic requests to servers. For example, search engines use robots (which are sometimes known as Web crawlers) to retrieve pages for inclusion in their search database. You can provide a robots.txt file to identify URLs that robots are not allowed to visit.

About this task

On visiting a Web site, a robot should make a request for the document robots.txt, using the URL

`http://www.example.com/robots.txt`

where `www.example.com` is the host name for the site. If you have host names that can be accessed using more than one port number, robots should request the robots.txt file for each combination of host name and port number. The policies listed in the file can apply to all robots, or name specific robots. Disallow statements are used to name URLs that the robots should not visit. Note that even when you provide a robots.txt file, any robots which do not comply with the robots exclusion standard might still access and index your Web pages.

If a Web browser requests a robots.txt file and you do not provide one, CICS sends an error response to the browser as follows:

- If you are using the CICS-supplied default analyzer DFHWBAAX, a 404 (Not Found) response is returned. No CICS message is issued in this situation.
- If you are using the sample analyzer DFHWBADX, or a similar analyzer which is only able to interpret the URL format that was required before CICS TS Version 3, the analyzer is likely to misinterpret the path `robots.txt` as an incorrectly specified converter program name. In this case, message DFHWB0723 is issued, and a 400 (Bad Request) response is returned to the browser. To avoid this situation, you can either modify the analyzer program to recognize the robots.txt request and provide a more suitable error response, or provide a robots.txt file using a URIMAP definition (which means that the sample analyzer program is bypassed for these requests).

To provide a robots.txt file for all or some of your host names:

Procedure

1. Create the text content for the robots.txt file. Information about creating a robots.txt file, and detailed examples, are available from several Web sites. Search on “robots.txt” or “robots exclusion standard” and select an appropriate site.

2. Decide how to store and provide the robots.txt file. You can provide the file using only a URIMAP definition, or using an application program.
   a. You can store the robots.txt file on z/OS UNIX System Services, and provide the file as a static response using a URIMAP definition. Most Web servers store the robots.txt file in the root directory for the host name. For CICS, a URIMAP definition can provide a file stored anywhere on z/OS UNIX, and the same file can be used for more than one host name.

      If you use a file stored on z/OS UNIX, the CICS region must have permissions to access z/OS UNIX, and it must have permission to access the z/OS UNIX directory containing the file, and the file itself. *Java Applications in CICS* explains how to grant these permissions.
   b. You can make the robots.txt file into a CICS document, and provide it either as a static response using a URIMAP definition, or as a response from an application program. The *CICS Application Programming Guide* explains how to create a CICS document template. A document template is defined using a DOCTEMPLATE resource definition, and it can be held in a partitioned data set, a CICS program, a file, a temporary storage queue, a transient data queue, an exit program or a z/OS UNIX System Services file.
   c. If you want to provide the contents of the robots.txt file using an application program, create a suitable Web-aware application program. Chapter 6, "Writing Web-aware application programs for CICS as an HTTP server," on page 75 tells you how to write an application program that uses the EXEC CICS WEB API commands. For example, you can use the EXEC CICS WEB SEND command with the FROM option to specify a buffer of data containing your robots.txt information. Alternatively, you can use the application program to deliver a CICS document from a template. Specify a media type of `text/plain`.

      You might want to use an application program to handle requests from robots so that you can track which robots are visiting your Web pages. The User-Agent header in a robot's request should give the name of the robot, and the From header should include contact information for the owner of the robot. Your application program could read and log these HTTP headers.
3. Begin a URIMAP definition that matches requests made by Web robots for the robots.txt file. "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100 lists the steps to create a URIMAP resource definition matching a request. The following sample URIMAP definition attributes could be specified to match a request for a robots.txt file for any host name:

```
 Urimap       ==> robots         - URIMAP name
 Group        ==> MYGROUP        - Any suitable
 Description  ==> Robots.txt
 STatus       ==> Enabled
 USAge        ==> Server         - For CICS as HTTP server
UNIVERSAL RESOURCE IDENTIFIER
 SCheme       ==> HTTP           - Will also match HTTPS requests
 HOST         ==> *              - * matches any host name.
                                        Specify host name if you
                                        provide separate robots.txt files
 PAth         ==> /robots.txt    - Robots use this path to
                                        request robots.txt
ASSOCIATED CICS RESOURCES
 TCpipservice ==>                - Blank matches any port. Specify
                                        TCPIPSERVICE definition name if
                                        you provide different robots.txt
                                        files depending on the port
```

Remember that the path components of URLs are case-sensitive. The path `/robots.txt` must be specified in lower case.

4. If you are providing the robots.txt file as a static response, complete the URIMAP definition to specify the file location and the other information which CICS Web support needs to construct responses. "Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server" on page 103 guides you through this process. For example, the following URIMAP definition attributes could be specified to provide a robots.txt file which was created using the EBCDIC code page 037 and stored in the `/u/cts/CICSHome` directory:

```
STATIC DOCUMENT PROPERTIES
 Mediatype    ==> text/plain
 CHaracterset ==> iso-8859-1
 HOSTCodepage ==> 037
 HFsfile      ==> /u/cts/CICSHome/robots.txt
```

   The HFsfile name is case-sensitive.

5. If you are providing the content of the robots.txt file using an application program, complete the URIMAP definition to specify that the program will handle requests. "Completing a URIMAP definition for an application response to an HTTP request for CICS as an HTTP server" on page 102 guides you through this process. For example, the following URIMAP definition attributes could be used to make the Web-aware application program ROBOTS handle the request, with no analyzer or converter program involved:

```
ASSOCIATED CICS RESOURCES
 Analyzer     ==>No             - Analyzer not used for request
 COnverter    ==>             - Blank means no converter program
 TRansaction  ==>             - Blank defaults to CWBA
 PRogram      ==> ROBOTS
```

Warning 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 is written to the CWBW transient data queue.

The message number used to record a warning header on a request (for CICS as an HTTP server) is DFHWB0750, and for a warning header on a response (for CICS as an HTTP client) it is DFHWB0752. The message for each warning header contains:

- The text associated with the warning header.
- The IP address of the server and client.

The messages are written to the CICS-supplied transient data queue CWBW, which is indirected to CSSL. There is a sample definition for the queue in group DFHDCTG.

CWBO is the queue normally used for CICS Web support messages, and CWBW is provided to keep warning messages separate. If you receive too many warning headers, or warning headers that are no longer useful (such as a warning that is always sent by a server in response to a client request that you make repeatedly), you can remove the CWBW transient data queue to suppress these records.

Chapter 9. Web error program

When a request error or an abend occurs in the CICS Web support process, a user-replaceable Web error program provides an error response to the Web client.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

A Web error program is used in the following situations:

- When CICS Web support detects a problem in initial processing of a request from a Web client; for example, if required information is missing from the request, or if the request is sent too slowly and the receive timeout is reached.
- When an installed URIMAP definition matches the request, but the URIMAP definition or virtual host is disabled, or the resource for a static response cannot be accessed.
- When URIMAP matching fails, and the analyzer specified for the TCPIPSERVICE definition is unable to process the request and passes control to a Web error program.
- When neither the URIMAP definition, nor the analyzer and converter program processing, manages to determine what application program should be executed to service the request.
- When an abend occurs in the analyzer program, converter program, or user-written application program. This ensures that a response can be returned to the Web client even though processing has failed.

A Web error program is **not** used in the following situations:

- When a sockets send or receive error occurs. In this case, the socket is closed and no response is sent to the Web client.
- When a URIMAP specifies a redirection response. These responses are composed by CICS and are not customizable.
- When a user-written application program has completed processing successfully and wants to return a response indicating an error, for example, if the client has specified a method not supported for the resource. These responses are composed and sent by the application.
- For processing involving CICS as an HTTP client. Web clients are not required to send an error response to servers. Responses received from servers are handled by the client application program.

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.

Two user-replaceable Web error programs are provided with CICS:

DFHWBERX (Web error application program)
: DFHWBERX can be specified as an application program to handle a request, by an analyzer program or in a URIMAP definition. It is used when the 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. DFHWBERX uses the EXEC

CICS WEB and DOCUMENT API commands to obtain information about the Web client's request and create and send the error response.

DFHWBEP (Web error program)
DFHWBEP is used in all other situations where a Web error program is required, that is, when CICS detects an error in request processing. CICS provides DFHWBEP with a parameter list giving information about the error situation, and a default error response in a block of storage. DFHWBEP can use the EXEC CICS WEB and DOCUMENT API commands to create its own error response and send it to the Web client, or it can amend or accept the default error response provided by CICS.

For more information about writing user-replaceable programs, see Customizing with user-replaceable programs in the *CICS Customization Guide*.

DFHWBERX, Web error application program

DFHWBERX uses the **EXEC CICS WEB** and DOCUMENT API commands to obtain information about the Web client's request and create and send the error response. It is called as an application program. DFHWBERX can be specified by an analyzer program, or as the PROGRAM attribute in a URIMAP definition if an error response is always wanted for the request.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

DFHWBERX is used when the 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. DFHWBAAX sets DFHWBERX as the application program to handle the request, using the `wbra_server_program` output parameter.

DFHWBERX is user-replaceable. CICS supplies the source code for DFHWBERX in Assembler only.

DFHWBERX does not receive a parameter list or a default HTTP response from CICS. It uses **EXEC CICS** commands to obtain information about the Web client's request and create and send the error response.

DFHWBERX provides an error response as follows:

- If the Web client's 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.

In DFHWBERX:

- The **EXEC CICS WEB EXTRACT** command is used to obtain the URL of the Web client's request for which an error response is needed.
- **EXEC CICS DOCUMENT** commands are used to construct the message body.
- For SOAP fault responses, the **EXEC CICS WEB WRITE HTTPHEADER** command is used to write an appropriate SOAP action header.

- The **EXEC CICS WEB SEND** command is used to specify an appropriate status code and send the response to the Web client. The UTF-8 character set is specified for code page conversion of the response body.

DFHWBERX does not use the **EXEC CICS WEB RECEIVE** command to receive the content of the Web client's request. If you are replacing DFHWBERX with your own application program, note that this command should not be used in Web error programs. If you are using the CICS-supplied default analyzer DFHWBAAX, DFHWBERX is used to send an error response to any request that is not matched by a URIMAP definition. The content of these requests cannot be known, and their intent could potentially be malicious, so it is not advisable to attempt to receive the request.

DFHWBEP, Web error program

DFHWBEP receives a parameter list from CICS giving information about the error situation, and a block of storage containing the default HTTP response (including status code and status text) that CICS plans to send to the Web client. The program can use or modify the default response, or create and send its own response using the EXEC CICS WEB and DOCUMENT API commands. DFHWBEP is user-replaceable.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Assessing the error situation

The parameter list passed to DFHWBEP by CICS contains the three-digit HTTP status code that CICS has used in the default response. The parameter list also supplies information that identifies the error situation, such as an error code, abend code, CICS message number, response and reason codes, and the name of the program where the error has occurred.

If you customize DFHWBEP, make sure that you are 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. Each status code may be used by CICS Web support for a variety of purposes. Any HTTP responses with status codes that are not known to your program should be passed through unchanged.

In addition to examining the parameter list provided by CICS, you might want to use the EXEC CICS WEB EXTRACT command and the EXEC CICS EXTRACT TCPIP command, to examine the request line and obtain other information relating to the Web client's request for which the error response is needed. The WEB READ HTTPHEADER command or the HTTPHEADER browsing commands can also be used to read the HTTP headers for the request, although you should be aware that these might not be available if the request was in an invalid state or timed out.

However, note that the EXEC CICS WEB RECEIVE command (which receives the content of the Web client's request) should **not** be used in Web error programs. In the range of error situations handled by DFHWBEP, the Web client's request might have timed out, or it might be lacking required information, or it might have unanticipated and potentially malicious content, or it might have already been received by another application program. Receiving a request which is in any of these states can lead to problems or unpredictable results, so it is not advisable to attempt to receive a Web client's request in your Web error program.

Creating and sending the error response

The parameter list provided by CICS includes a pointer to a block of storage containing the default HTTP response for the error detected, and also a parameter giving the length of the response. The block of storage contains a complete HTTP response, including the status line, HTTP headers and message body, and the length is the length of the complete response message.

The Web error program can choose one of the following actions:

1. Leave the default response unchanged and allow CICS to send it to the Web client. Take this action for any HTTP responses with status codes that are not known to your program, or in cases where you have assessed the situation and found that the default response is appropriate.
2. Use the **EXEC CICS** WEB and DOCUMENT API commands to create a new response and send it to the Web client. This action is recommended if you want to change the response, because it means that CICS can provide more assistance with checking the message. A response that you create in this way replaces the default response, which is discarded. The WEB WRITE HTTPHEADER command can be used to write HTTP headers for the response, and the WEB SEND command can be used to assemble and send the response. You must specify ACTION(IMMEDIATE) in your command, as the default of ACTION(EVENTUAL) is not permitted with DFHWBEP. "Writing HTTP headers for a response" on page 84, "Producing an entity body for an HTTP message" on page 86, and "Sending an HTTP response from CICS as an HTTP server" on page 86 explain how to create and send a response using the API commands.
3. Modify the default response manually in the block of storage, update the length parameter accordingly, and allow CICS to send it to the Web client. This action could be considered if you only want to make minor changes to the default response (such as replacing the default message body with a short piece of text), but you must be careful to ensure that the HTTP response remains valid and that the correct length is stated.
4. Construct a new HTTP response manually in a new block of storage, pass back the address of the new block of storage and the length of the new response, and allow CICS to send it to the Web client. The new response replaces the default response, which is discarded. This action is no longer recommended, because CICS cannot provide full assistance with checking a message constructed in this way. If you have a version of DFHWBEP which was customized before CICS Transaction Server for z/OS, Version 3 Release 2 and takes this action, you should consider replacing this action with an HTTP response constructed and sent using the **EXEC CICS** WEB and DOCUMENT API commands.

Correct content for the error response

Whether you decide to use the **EXEC CICS** WEB and DOCUMENT API commands to create a new response, or modify the default response manually in the block of storage, or construct a new HTTP response manually in a new block of storage, it is possible to modify all the items in the error response. However, you must be careful to ensure that the HTTP response remains valid and appropriate, and if you are working with the response manually in a block of storage, that the correct length is stated.

The response must contain an HTTP version, status code, status text, any HTTP headers that are required, and the message body. The format of the response

should be compliant with the HTTP protocol specification to which you are working (HTTP/1.0 or HTTP/1.1). If you are using the API commands, CICS provides assistance with all these elements.

Note the following guidance for individual items in the error response:

The HTTP version (HTTP/1.1 or HTTP/1.0)
In the default response, this is decided by CICS according to the HTTP version of the Web client. If you are working with the default response in the block of storage, do not modify this element of the response. If you are creating a new response using the API commands, use the WEB EXTRACT command to identify the HTTP version of the Web client, and tailor your response accordingly. The HTTP version used by the Web client can affect your choice of HTTP headers, status code, and message content for the response. HTTP/1.0 clients might not understand the more advanced features described in the HTTP/1.1 specification.

The numeric status code (for example, 404 or 500)
CICS chooses a status code for the default response. Be cautious when modifying this element of the default response or choosing a status code for your new response. Appendix C, “HTTP status code reference for CICS Web support,” on page 381 lists the status codes used by CICS Web support, and the reasons why they are used. The HTTP/1.1 specification has more information about all the status codes and the requirements for their correct use. If you decide that a different status code is more appropriate than the one selected by CICS Web support, make sure your usage is compliant with the requirements in the HTTP/1.1 specification. In particular, check that the status code is suitable for the HTTP version of the Web client. For non-HTTP errors, CICS always uses a 400 status code.

The reason phrase, or status text (for example, Not Found)
You may modify this element of the default response, or supply your own reason phrase for a new response. The reason phrases suggested by the HTTP/1.1 specification (for example, "Not Found" or "Bad Request") are recommended but optional. The HTTP/1.1 specification says that the reason phrases for each status code may be replaced by local equivalents.

HTTP headers
The default response contains the headers that CICS has written for the response (for example, Date and Server headers). If you are creating a new response using the API commands, CICS adds these headers automatically when you send the response. Appendix B, “HTTP header reference for CICS Web support,” on page 375 lists the headers that CICS can write. The headers written by CICS are appropriate for the HTTP version of the message, and should not be removed if you are working with the default response, because they might be required for compliance with the HTTP specifications. You may add further HTTP headers for the response if appropriate. Check that the HTTP/1.1 specification allows the use of the headers in that context. If you have selected a different status code, some headers may be required by the HTTP/1.1 specification.

Message body
The message body for the default response repeats the status code and reason phrase that are given in the release line. You may modify this element of the default response, or supply your own message body for a new response. For many status codes, the body of the message may be used to provide further information to a human user. Some status codes cannot be accompanied by a message body.

Code page conversion

The default HTTP response in the block of storage is passed to DFHWBEP in the EBCDIC code page 037.

When you use **EXEC CICS** WEB API commands in the Web error program to produce a new error response and send it to the Web client, code page conversion takes place as you specify in the commands, in the same way as for any other program which uses the **EXEC CICS** WEB API commands.

DFHWBEP is not able to specify code page conversion settings for a response produced in a block of storage. If you modify the default error response in the block of storage, or supply a new error response in a new block of storage, and return this to CICS for sending, CICS assumes that the response provided in the block of storage is also in the EBCDIC code page 037.CICS performs code page conversion on the response to convert it to a suitable ASCII character set before returning it to the client. If an analyzer program is involved in the processing path and has set parameters for code page conversion (as individual server and client code page parameters, or as a DFHCNV key), CICS uses these options for code page conversion. If no analyzer program is involved, or the analyzer was not invoked before the error occurred, the ISO-8859-1 character set is used for the response. If this outcome is not suitable, use the **EXEC CICS** WEB API commands to produce the response instead.

Input parameters for DFHWBEP, Web error program

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS passes several parameters to DFHWBEP, including an error code, an abend code and an error message.

For a listing and technical descriptions of all the parameters in the list passed to DFHWBEP by CICS, see Appendix H, “Reference information for DFHWBEP, Web error program,” on page 421.

The input parameters for DFHWBEP are as follows:

- An error code identifying the cause of the original error (wbep_error_code). The DFHWBUCO copybook lists these codes.
- The type of processing that was in progress when the error occurred (server or pipeline).
- The name of the program in which the error occurred.
- The CICS abend code associated with the error (wbep_abend_code).
- The CICS message number associated with the error and a pointer to the message text (wbep_message_number).
- The response and reason codes returned by the analyzer or the converter program, if used.
- The name of the TCPIPSERVICE resource definition for the port on which the request was received. (This name identifies the name of the analyzer program, if used.)
- The name of any converter program that was used.
- The name of the target program, that is; the application that was intended to provide a response to the request.

- The 3-digit HTTP status code in the default HTTP response (wbep_http_response_code). For a list of the status codes used by CICS Web support, and the reasons why they are used, see Appendix C, "HTTP status code reference for CICS Web support," on page 381.
- A pointer to a block of storage containing the default HTTP response. This response is a complete HTTP response, including the status line, HTTP headers and message body.
- The length of the default HTTP response. The maximum length of the response buffer is 32 KB.
- The IP address, expressed in dotted decimal or colon hexadecimal notation, for CICS as an HTTP server for this request.
- The IP address, expressed in dotted decimal or colon hexadecimal notation, of the Web client.

For more information on IP address notation, see "IP address formats" on page 7.

Output parameters for DFHWBEP, Web error program

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The output parameters for DFHWBEP are `wbep_response_ptr` and `wbep_response_len`.

Appendix H, "Reference information for DFHWBEP, Web error program," on page 421 has a listing and technical descriptions of all the parameters in the in the list passed to DFHWBEP by CICS.

The output parameters for DFHWBEP are:

- The address of a block of storage containing an HTTP response (`wbep_response_ptr`).
- The length of the HTTP response in the block of storage (`wbep_response_len`). The maximum length of the response is 32K. The specified length is the length of the whole buffer; CICS calculates the length of the message body and supplies an appropriate Content-Length header.

By default, these output parameters relate to the block of storage containing the default HTTP response produced by CICS.

- If you have used the **EXEC CICS** WEB and DOCUMENT API commands in DFHWBEP to create a new response and send it to the Web client, CICS ignores and discards the HTTP response in the block of storage, so the output parameters can be left unchanged.
- If you have modified the default response in the block of storage, you need to update the length in `wbep_response_len`, giving the new length of the whole buffer. You do not need to calculate the message body length or change the Content-Length header in the response. CICS checks the length of the message body that you have provided, and corrects the Content-Length header accordingly.
- If you have constructed a new HTTP response manually in a new block of storage, you need to pass back the address of the new block of storage in `wbep_response_ptr` and the length of the new response in `wbep_response_len`.

CICS Web support default status codes and error responses

The response code and reason code set by an analyzer or converter program map to default status codes and associated responses. CICS also selects a default status code and associated response if an error occurs when a static response is produced using a URIMAP definition. The status code and response can be modified by the user-replaceable Web error program DFHWBEP.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The HTTP protocol specifications define status codes that a server can return for the HTTP response when a request cannot be completed successfully. Appendix C, "HTTP status code reference for CICS Web support," on page 381 gives information about these status codes.

For more information about the structure of HTTP responses, see "HTTP responses" on page 14.

When an error occurs during CICS Web support processing, information is passed to the Web error program DFHWBEP in a parameter list to assist in determining an appropriate error response:

- If an error occurs during processing by an analyzer or converter program, you can identify the error using the response and reason codes from the program in the parameter list.
- If an error occurs in producing a static response using a URIMAP definition, you can identify the error using the associated CICS message number and text in the parameter list.

For all types of error, a complete default error response, including the status code, is passed to the Web error program to be accepted, modified or replaced. Error responses are accompanied by a CICS message and an exception trace entry.

The default status code for response codes used by an analyzer program is as follows:

Table 3. Default status code for analyzer program processing error

wbra_response	Default status code
any value other than URP_OK	400 Bad Request

The default status codes for a converter program are as follows:

Table 4. Default status codes for the converter's decode function

decode_response	decode_reason	Default status code
URP_EXCEPTION	URP_CORRUPT_CLIENT_DATA	400 Bad Request
URP_EXCEPTION	URP_SECURITY_FAILURE	403 Forbidden
URP_EXCEPTION	any other value	501 Not Implemented
URP_INVALID	any value	501 Not Implemented
URP_DISASTER	any value	501 Not Implemented
any other value	any value	500 Internal Server Error

Table 5. Default status codes for the converter's encode function

encode_response	encode_reason	Default status code
Any value other than URP_OK URP_OK_LOOP	any	501 Not Implemented

The default status codes for errors in producing a static response using a URIMAP definition are as follows:

Table 6. Default status codes for static response processing errors (using a URIMAP definition)

CICS message number	Error	Default status code
0758	User does not have READ access to the resource needed to produce the static response (CICS document template or z/OS UNIX file).	403 Forbidden
0759	The resource needed to produce the static response cannot be found (CICS document template or z/OS UNIX file).	404 Not Found
0760	The z/OS UNIX file needed to produce the static response cannot be read.	500 Internal Server Error
0761	Any other error.	500 Internal Server Error

Chapter 10. Analyzer programs

Analyzer programs are associated with TCPIPSERVICE definitions. Their primary role is to interpret an HTTP request if a URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

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. The role of analyzer programs in the CICS Web support process for CICS as an HTTP server is described in "HTTP request and response processing for CICS as an HTTP server" on page 26. Chapter 5, "Planning your CICS Web support architecture for CICS as an HTTP server," on page 59 has information to help you plan your architecture for CICS as an HTTP server.

Relationship between analyzer programs and URIMAP definitions

Before CICS Transaction Server for z/OS, Version 3 Release 1, all HTTP requests for CICS as an HTTP server were interpreted by an analyzer program. In CICS TS Version 3, URIMAP definitions are the strategic facility to control the processing of HTTP requests. They replace key functions of the analyzer program in matching the URLs of requests to the application program that processes them, and specifying the use of a converter program and an alias transaction.

URIMAP definitions may, however, invoke an analyzer program for selected HTTP requests, to take over some of the processing stages, and to perform other actions such as monitoring or audit actions. The attributes of the URIMAP definition that reproduce analyzer functions, namely CONVERTER (converter program name), TRANSACTION (alias transaction), USERID (user ID for alias transaction), and PROGRAM (name of application program to process request), can be passed to the analyzer program, and the analyzer program can choose to override these.

You can choose to pass an HTTP request directly to an analyzer program without using a URIMAP definition, following the same process that CICS Web support used before CICS Transaction Server for z/OS, Version 3 Release 1. However, without URIMAP definitions, if you want to change the way in which CICS responds to a particular HTTP request, you need to change the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system management task. Also note that if you continue to use an analyzer program instead of a URIMAP definition to handle requests, and you need to be compliant with HTTP/1.1 in this respect, you must code your analyzer program to perform URL comparison according to the rules stated in the HTTP/1.1 specification (RFC 2616).

Note: As supplied, the CICS-supplied sample analyzer program DFHWBADX and the CICS-supplied default analyzer program DFHWBAAX do 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).

Use of analyzer programs for error handling

Although an analyzer program is not now required in the processing path for every HTTP request, an analyzer program must still be specified for each TCPIPSERVICE resource definition that is used for CICS Web support.

The name of the analyzer program is specified in the URM attribute of the resource definition. You can specify a different analyzer in each TCPIPSERVICE definition, or you can specify the same analyzer in more than one TCPIPSERVICE definition. If you are invoking an analyzer program from a URIMAP definition, you cannot choose between different analyzer programs; you can only select whether or not to use the analyzer program specified for the TCPIPSERVICE definition.

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 a Web error program, not the analyzer program.)

Because of this, as a minimum, the analyzer program specified for each TCPIPSERVICE definition should include a procedure to handle any HTTP request that it does not recognize, and provide a suitable error response. You may also identify specific requests that should have been handled by a URIMAP definition, and provide a more relevant error response. The output from an analyzer program in an error situation is passed to a Web error program, which you can use to modify the HTTP response. Chapter 9, “Web error program,” on page 115 explains how to tailor these.

The CICS-supplied default analyzer program DFHWBAAX is the default when a TCPIPSERVICE definition specifies PROTOCOL(HTTP). DFHWBAAX provides basic error handling when all requests on the port should be handled by URIMAP definitions. It does not provide support for requests using the URL format that CICS Web support used before CICS TS 3.1. If you need to provide handling in your analyzer program for requests that are not handled by URIMAP definitions, the analyzer program specified on your TCPIPSERVICE definition should be the CICS-supplied sample analyzer program DFHWBADX or your own customized analyzer program.

Use of analyzer programs for some non-Web-aware applications, and for non-HTTP messages

Non-Web-aware applications might function correctly when they are invoked directly from a URIMAP definition. However, some might be dependent on facilities that can only be provided for them by an analyzer program. The use of an analyzer program in the processing path for an HTTP request might be needed in the following circumstances:

- You are producing a response using a non-Web-aware application and a converter program, and it needs to be flagged for pre-CICS TS Version 3 compatibility processing, because a Web client requires a response identical with the response it would have received before CICS TS Version 3. (For example, user-written clients could experience problems with new error responses or additional HTTP headers.) This flag only works if the converter program

produces the response manually in a block of storage. If the converter program uses the **EXEC CICS** WEB API commands to send the response, the flag has no effect.

- You are producing a response using a non-Web-aware application and a converter program, and either the copy of the Web client's request which is passed to the converter program in a block of storage, or an HTTP response which the converter program produces manually in a block of storage, requires nonstandard code page conversion. A converter program is not able to specify code page conversion settings for HTTP requests or responses that are passed in a block of storage. The standard settings used by CICS for code page conversion if no analyzer program is present in the processing path are described in "Writing a converter program" on page 140. If these standard settings are not suitable, or if code page conversion is not wanted, you can use an analyzer program in the processing path to specify alternative code page conversion settings. As an alternative to using an analyzer program, you could use the **EXEC CICS** WEB API commands in the converter program to examine the Web client's request or to produce the response, instead of using the block of storage. In this case, code page conversion can be specified as usual on the **EXEC CICS** WEB API commands.

If you require an analyzer program to handle one of these situations, a URIMAP definition may be set up for the request, but it must specify the analyzer program.

For non-HTTP requests, which use the user-defined (USER) protocol on the TCPIPSERVICE definition, an analyzer program is always required to process the requests, and URIMAP definitions cannot be used. Chapter 14, "CICS Web support and non-HTTP requests," on page 187 explains how non-HTTP requests are processed.

Use of analyzer programs for additional processing

In situations where the use of an analyzer program in the processing path is optional, you might choose to use an analyzer program for reasons such as the following:

- You want to make dynamic changes to elements of the processing path, based on the content of the request. Each URL for a HTTP request is matched by a single URIMAP definition, which defines a single processing path. An analyzer program can interpret the content of the request and change elements such as the application program that handles the request, the involvement of a converter program, or the alias transaction and user ID used for the request.
- You want to introduce monitoring or audit actions into the process. An analyzer program is an appropriate location in which to do this.
- You are upgrading an existing CICS Web support architecture from CICS TS Version 2, and your existing analyzer program provides additional functions that you want to maintain during request processing, such as passing information to a converter program.

"Writing an analyzer program" on page 128 explains the full range of functions that an analyzer program can perform.

Replacing analyzer programs with URIMAP definitions

You should usually be able to replace the request processing functions of your 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, alias transaction and user ID. If your analyzer program provides 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:

- 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.
- 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.
- Ensure that your analyzer program still contains basic handling procedures for unrecognized requests, even if it is no longer involved in the processing path for any requests. The analyzer program is still required on the TCPIPSERVICE definition, and receives requests in situations such as the end user mis-typing a URL.

Writing an analyzer program

You can write an analyzer program in Assembler, C, COBOL, or PL/I.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Input and output parameters for an analyzer program are passed in a COMMAREA. Language-dependent header files, include files, and copy books which map the COMMAREA are described in Appendix E, “Reference information for analyzer programs,” on page 395.

The full range of functions which an analyzer program can perform is as follows:

- Determine whether processing should continue for the request, or whether CICS should return an error response to the Web client.
- Analyze the content of the request, and any parameters that have been passed to the converter program from a URIMAP definition, to determine which of the subsequent processing stages are required, and which CICS resources are needed to carry out each stage. (The **EXEC CICS** WEB API commands may be used during this analysis.)
- Specify the name of a converter program to process the request before it is passed to an application program. Converter programs are normally used with application programs that are not Web-aware. A user token is provided for the analyzer program to communicate with the converter program, if required. The Web client's request is passed to the converter program in a 32K block of storage indicated by a pointer in the parameter list. Chapter 11, “Converter programs,” on page 139 explains the functions of a converter program.
- Specify the name of the user-written application program that is to process the request and provide the response.

- Specify the transaction ID of the alias transaction that handles the remaining stages of processing.
- Specify a user ID that is to be associated with the alias transaction.
- Specify or suppress code page conversion for the request passed to the converter program in the block of storage, and any response that the converter program constructs manually in a block of storage. This does not affect converter programs or user-written applications which use the **EXEC CICS** WEB API commands to view the HTTP request and produce the response; they request code page conversion directly from CICS. "Code page conversion for CICS Web support" on page 39 explains the code page conversion process.
- Specify the flag, provided for upgrade purposes, that indicates where a non-Web-aware application requires pre-CICS TS Version 3 compatibility processing. This does not affect converter programs or user-written applications which use the **EXEC CICS** WEB API commands to view the HTTP request and produce the response.
- Modify the request body. Any changes made are visible in the data passed to the converter program in the block of storage, but **not** to the **EXEC CICS** WEB API commands.

CICS supplies the default analyzer program DFHWBAAX, which is described in "CICS-supplied default analyzer program DFHWBAAX" on page 135, and the sample analyzer program DFHWBADX, which is described in "CICS-supplied sample analyzer program DFHWBADX" on page 135. If these analyzers do not meet your requirements, you need to write your own. You might be able to use DFHWBADX as an example.

All the user-replaceable programs must be local to the system in which CICS Web support is operating. If you do not use autoinstall for programs, you must define and install program definitions for all user-replaceable programs used by CICS Web support, including the analyzer and converter programs. If you use autoinstall for programs, you must ensure that user-replaceable programs are installed with the correct attributes. Note that your analyzer programs must be defined with EXECKEY(CICS).

For more information about writing user-replaceable programs, see Customizing with user-replaceable programs in the *CICS Customization Guide*.

Input to an analyzer program

Input parameters are passed to the analyzer program in a COMMAREA, giving information about the nature and content of the request, and any input supplied by a URIMAP definition. The analyzer program can choose to accept these values and pass them on as output parameters, or it can dynamically override them based on its analysis of the content of the request.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

"Parameters for analyzer programs" on page 396 has a listing and technical descriptions of all the parameters in the COMMAREA.

The input parameters include the following items, or a pointer to them:

- An eye-catcher for an analyzer parameter list.
- The colon hexadecimal or dotted decimal IP address of the client and of the server (CICS as an HTTP server).

- An indicator of whether the request is an HTTP request.
- An indicator of whether a matching URIMAP definition was found for the request. If this indicator is positive, the URIMAP definition might have passed additional input parameters to the analyzer program.
- The HTTP version.
- The request method.
- The host name specified for the request, taken from the Host header or, for an absolute URI, from the request URL. For HTTP/1.1 requests, a host name is required, so this parameter is always passed to the analyzer. For HTTP/1.0 requests, a host name might not be supplied.
- The path component of the URL.
- Any query string that was specified for the request.
- The HTTP headers for the request.

  If the request has been sent using chunked transfer-coding, any trailing headers are not passed to the analyzer program with the main request headers.
- The request body, or as much of the request body as fits into a 32 KB block of storage. This request body is a pointer to the separate block of storage containing the request.

For HTTP requests received on a connection using SSL client authentication, the following parameter is also passed:

- The user ID obtained from the client certificate.

If a matching URIMAP definition is found for the request and it invoked the analyzer program, the following parameters from the URIMAP definition are passed to the analyzer program, if they are present in the URIMAP definition:

- The name of the recommended converter program to process the request before it is passed to an application program (CONVERTER attribute in the URIMAP definition).
- The name of the recommended user-written application program to process the request and provide the response (PROGRAM attribute in the URIMAP definition).
- The transaction ID of the recommended alias transaction to cover the remaining stages of processing (TRANSACTION attribute in the URIMAP definition).
- The recommended user ID that is to be associated with the alias transaction (USERID attribute in the URIMAP definition). This user ID can be overridden if a user ID is supplied by the client.

The **wbra_urimap** input parameter can be used to test whether or not a URIMAP definition was used in the processing path for the request.

If you are using an analyzer program instead of a URIMAP definition to handle requests, and you need to comply with HTTP/1.1 in this respect, you must code your analyzer program to perform URL comparison according to the rules stated in the HTTP/1.1 specification. Under these rules, scheme names and host names are compared case-insensitively, but paths are compared case-sensitively. All components are unescaped before comparison. When CICS compares URLs to URIMAP definitions, it follows these rules.

You can also use the **EXEC CICS** WEB API commands to examine the HTTP request, if preferred. Using the **EXEC CICS** WEB commands can increase the accuracy and completeness of your analysis of the request, particularly when examining the HTTP headers, which are subject to wide variation in content and usage. The **EXEC**

CICS WEB commands also simplify the process of locating and extracting query string or formfield information from a request, which can be a determine the subsequent processing.

You can use the EXTRACT TCPIP command to obtain the following information about the client request that is being processed:

- The IP address of the Web client
- The host name of the Web client, as known by the DNS server
- The number of the port on which the Web client sent its connection request
- The IP address of the server; that is, CICS as an HTTP server
- The type of authentication in use
- The level of SSL support in use
- The TCPIPSERVICE resource definition associated with the request

Output from an analyzer program

An analyzer program provides output in a COMMAREA. The output includes a response code, and a range of optional output parameters that can be used to specify further processing stages and to share information with a converter program.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

"Parameters for analyzer programs" on page 396 has a listing and technical descriptions of all the parameters in the COMMAREA.

The analyzer program must provide the following output in its COMMAREA:

- A response code.
  - If your analyzer program returns a response code of URP_OK, processing continues with the next step.
  - If your analyzer program returns any other value, CICS returns an error response to the Web client. The response can be modified with a user-replaceable Web error program. "CICS Web support default status codes and error responses" on page 122 tells you how the return codes from the analyzer map to the status codes that CICS returns to the Web client.

The analyzer program may also provide the following outputs:

- The name of a converter program that is to be used to process the request before it is passed to a user-written application program.
  - If a converter program name was input from a URIMAP definition, you can accept or override this.
  - If the analyzer indicates that a converter program is not required, the first 32K bytes of the request is passed to the user-written application program in a block of storage. A Web-aware application can ignore this and use the **EXEC CICS** WEB API commands to read the request.
- The name of an application program that is to process the request and provide the response.
  - If a program name was input from a URIMAP definition, you can accept or override this.
  - If you are using a converter program, the converter program can specify or override the program name. A converter can be used in this way to involve more than one program in processing the request.

- The transaction ID of the alias transaction that is to cover the remaining stages of processing. If a transaction ID was input from a URIMAP definition, you can accept or override this.
- The user ID that is to be associated with the alias transaction. If a user ID was input from a URIMAP definition, you can accept or override this. This is how CICS determines the user ID if you do not specify one:
  - If a user ID was input from a URIMAP definition, that is used.
  - If the HTTP request uses SSL with client authentication, the user ID is obtained from the client certificate.
  - In other cases, the CICS default user ID is used.
- Parameters relating to code page conversion of the 32K block of storage containing the request, and to code page conversion of the response body, if the converter program produces it manually in a block of storage.

  Note: This does not affect converter programs or user-written applications which use the **EXEC CICS** WEB API commands to view the HTTP request and produce the response; they request code page conversion directly from CICS. You can specify the parameters for conversion of the block of storage containing the request in one of two ways:
  - As a pair of parameters specifying the character set used by the Web client (`wbra_characterset`), and the host code page suitable for the application program (`wbra_hostcodepage`). Specifying the parameters in this way means that an entry in the code page conversion table (DFHCNV) is not required.
  - As a key for an entry in the DFHCNV code page conversion table (`wbra_dfhcnv_key`). This is not recommended, except for upgrade purposes.

  If you do not specify any of these parameters, the default behavior is for CICS to convert a text message using the standard settings described in Chapter 10, "Analyzer programs," on page 125. If you want to suppress code page conversion for the request and response in the block of storage, set `wbra_dfhcnv_key` to nulls or blanks.
- The flag that indicates where a non-Web-aware application that uses a converter program requires pre-CICS TS Version 3 compatibility processing (wbra_commarea). This flag is provided for upgrade purposes. It can be used only by applications that do not use the **EXEC CICS** WEB API commands (that is, they produce the response manually in a block of storage), 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 of the response headers that are normally inserted for HTTP/1.1 messages. Only the headers that were sent to clients before CICS Transaction Server for z/OS, Version 3 Release 1 are used.
  - 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 an HTTP/1.1 response.
- An eight byte user token, used to share information between the analyzer and converter programs. "Sharing data between analyzer and converter programs" on page 133 explains how this works.
- A modified value for the request body length.

The analyzer can modify the contents of the request:

- The modified data can be shorter than, or of the same length as the original data. The request body cannot be lengthened.
- Any changes made are visible in the data passed to the converter program, but **not** to the **EXEC CICS** WEB API commands.

Sharing data between analyzer and converter programs

CICS passes three parameters between the analyzer and the converter programs that enable data to be shared by these processing stages.

The user_data pointer
This parameter contains the address of a 32K block of storage that is passed from stage to stage. On entry to the analyzer program, the pointer points to a block of storage containing the HTTP request. On completion of the encode function of the converter program, CICS Web support uses it to locate the block of storage containing the HTTP response, unless the **EXEC CICS** WEB API commands have been used to produce a response instead.

You must not change the value of the pointer in the analyzer program, although you can modify the contents of the block of storage addressed by the pointer.

Between the converter program and the user-written application program, you can pass the pointer unchanged from one stage to another, or you can issue a GETMAIN command in one program and pass the address of the newly acquired storage in the pointer.

The user_data length
This parameter is the length of the block of storage addressed by the user_data pointer.

The user token
The user token is an 8–byte field which is shared by the analyzer program and the converter program. It can contain any information you want:

- You can pass small quantities of shared information directly in the user token.
- To pass larger quantities, you can issue a GETMAIN command in one program, to acquire storage for a shared work area. Use the user token to pass the address of the shared storage.

You can change the contents of the user token in each program: for example, the user token can have one meaning when passed from the analyzer program to the decode function of the converter program, and a different meaning when passed to the encode function.

The analyzer program can modify any of the parameters in the parameter list which is passed to the converter program. The pointers cannot be changed, but the data indicated by the pointer can be changed. The length of each field must not change.

Note: The analyzer and converter programs execute under different CICS tasks. Therefore, if you issue a GETMAIN command in the analyzer program, you must code the SHARED option if the storage is to be visible in the converter program. In general, storage acquired with the SHARED option is not freed automatically by CICS, so you must issue a GETMAIN command when your programs no longer need the storage. However, CICS will free the storage addressed by the user_data pointer after the HTTP response has been sent to the Web client.

Selecting escaped or unescaped data from an analyzer program

The HTTP request which is passed to the analyzer program for parsing is in its escaped form. Reserved or excluded characters in the URL, or in form data in the message body, are presented as a %xx sequence, wherexx is the ASCII hexadecimal representation of the reserved character. The analyzer can pass the request in a 32K block of storage to subsequent processing stages in its escaped form, with the escape sequences still present, or in its unescaped form, with the escape sequences converted back to the original characters. Web-aware application programs using the **EXEC CICS** WEB API commands do not use this mechanism to receive the response, and they request unescaping directly from CICS.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

"Escaped and unescaped data" on page 16 explains escaping and its purpose. Escaping and unescaping only applies to the following elements of the HTTP request:

- The URL portion of the request line, including any query string. The query string might be data from a form with the GET method.
- Form data returned from a form with the POST method and the default encoding **application/x-www-form-urlencoded**. This data is presented in the message body."HTML forms" on page 17 explains more about form data.

If the request in the 32K block of storage is to be passed on in unescaped form, the analyzer can convert the data from escaped to unescaped form, or have CICS perform the conversion.

- To pass the request in escaped form, set WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED in your analyzer. WBRA_UNESCAPE_NOT_REQUIRED is the default value.
- To pass the request in unescaped form and have CICS perform the conversion, set WBRA_UNESCAPE to WBRA_UNESCAPE_REQUIRED in your analyzer.
- To pass the request in unescaped form after the analyzer has performed the conversion, set WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED.

Web-aware application programs using the **EXEC CICS** WEB API commands do not use the COMMAREA mechanism to receive and send the response, and they request unescaping directly from CICS. For Web-aware applications that use the **EXEC CICS** WEB API commands, when you extract form data from a request using the WEB READ FORMFIELD command or form field browsing commands, CICS performs the unescaping, and the data is returned in its unescaped form. When you extract a query string from a request using the WEB EXTRACT command, the data is returned in its escaped form.

If you are writing an application with a COMMAREA interface that can be run either through CICS Web support or through the CICS business logic interface, ensure that WBRA_UNESCAPE is set to WBRA_UNESCAPE_NOT_REQUIRED, and that any unescaping is delegated to the application. If this is not done, the application is passed unescaped data by the CICS business logic interface, and escaped data by CICS Web support, which might cause unpredictable results.

CICS-supplied default analyzer program DFHWBAAX

CICS supplies a default analyzer program, DFHWBAAX. DFHWBAAX provides an error handling function for TCPIPSERVICE resource definitions that are used for CICS Web support. It is suitable for use when all of the requests using a port are handled using URIMAP definitions.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS supplies the source code for DFHWBAAX in Assembler only.

DFHWBAAX is the default analyzer program for a TCPIPSERVICE definition that specifies PROTOCOL(HTTP).

DFHWBAAX receives the same input and output parameters as a standard analyzer program, in a COMMAREA. As supplied, it does not make use of most of these parameters, and it does not provide support for requests using the URL format that CICS Web support used before CICS TS 3.1. Instead, it takes simplified action as follows:

- DFHWBAAX does not carry out further processing when a matching URIMAP definition has been found for the request, even if the URIMAP specifies ANALYZER(YES). It uses the `wbra_urimap` input parameter to test for the presence of a URIMAP definition, and if the result is positive, returns without performing any analysis on the request URL. This means that the settings specified in the URIMAP definition for the alias transaction, converter program (if used), and application program are automatically accepted and used to determine subsequent processing stages.
- If no matching URIMAP definition is found, DFHWBAAX gives control to the user-replaceable Web error transaction program DFHWBERX to produce an error response. This is achieved by setting DFHWBERX as the application program to handle the request, using the `wbra_server_program` output parameter. DFHWBAAX does not make any other changes to the COMMAREA. On receiving control, DFHWBERX provides either an HTTP response with a 404 (Not Found) status code, or a SOAP fault response, depending on the request made by the Web client.

DFHWBAAX uses a standard range of responses, **URP_OK**, **URP_EXCEPTION**, and **URP_INVALID**. No reason values are architected for DFHWBAAX as supplied. Note that if the response is other than **URP_OK**, this indicates an error in processing, and control is passed to the user-replaceable Web error program DFHWBEP, rather than the Web error application program DFHWBERX.

CICS-supplied sample analyzer program DFHWBADX

CICS supplies a working sample analyzer program, DFHWBADX. If you need to provide request handling through your analyzer program, as well as or instead of through URIMAP definitions, you can use DFHWBADX as a starting point for writing your own analyzer program.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS supplies the source code in several languages:

- DFHWBADX (Assembler)

- DFHWBAHX (C)
- DFHWBALX (PL/I)
- DFHWBAOX (COBOL)

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, and if the result is positive, returns without performing any analysis on the request URL. If you write your own analyzer program and want it to interact with a URIMAP definition, do not copy this aspect of DFHWBADX's processing. You may want to test the `wbra_urimap` input parameter in order to modify your analyzer program's processing in other ways. For example, you could test the parameter to decide whether to perform analysis based on the input parameters from the URIMAP definition, or to perform analysis directly on the request URL.

How DFHWBADX interprets a request URL

DFHWBADX interprets HTTP requests in which the path component of the URL has the following syntax:

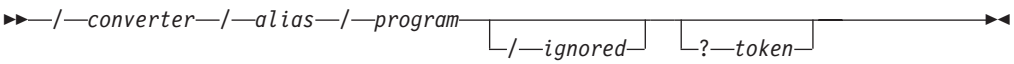


Figure 7. Syntax of path component interpreted by DFHWBADX

All fields processed by the analyzer program are translated to upper case. After translation:

converter
Specifies the name of the converter program to be used for the request. It can be up to eight characters in length.

As a special case, the four character value `'CICS'` denotes that no converter program is used. See Chapter 11, “Converter programs,” on page 139 for information on how to use converter programs with URIMAP definitions.

alias
Specifies the transaction ID of the alias transaction for subsequent request processing. It can be up to four characters in length.

program
Specifies the name of the CICS application program that is to be used to service the request. It can be up to eight characters in length.

ignored
This part of the path is ignored by DFHWBADX (but may be used by the converter program or the application program).

token
The first eight bytes specify the user token that is passed to the converter program. Data following the first eight bytes of the token is ignored by DFHWBADX (but may be used by the converter program or the application program).

In the example path /cics/cwba/dfh$wb1a:

- No converter program is used.
- The alias transaction is CWBA.
- The CICS application program is DFH$WB1A.

In addition to the outputs derived from the original HTTP request, DFHWBADX sets the following outputs:

- The code page conversion template is DFHWBUD. This template is defined in sample conversion table DFHCNVW$, and converts data between the ASCII Latin-1 character set (code page ISO 8859–1) and the EBCDIC Latin character set (code page 037). The sample conversion table can be used without any configuration, but note that the output parameters wbra_characterset and wbra_hostcodepage can be used in place of the wbra_dfhcnv_key output parameter to provide greater control and avoid the use of a conversion table.
- DFHWBADX passes the request in escaped form, and sets WBRA_UNESCAPE_NOT_REQUIRED.

Responses from DFHWBADX

The meanings of the responses produced by DFHWBADX are as follows:

URP_OK
The analyzer found that the request conformed to the default HTTP request format, and generated the appropriate outputs for the alias.

URP_EXCEPTION
The analyzer found that the request did not conform to the default format. A reason code is supplied as follows:

1 The length of the resource was less than 6. (With the URL format recognized by DFHWBADX, the shortest possible resource specification is /A/B/C, asking for program C to be run under transaction B with converter A.) This response and reason are the ones used when the incoming request is not an HTTP request.

2 The resource specification did not begin with a "/".

3 The resource specification contained one "/", but fewer than three of them.

4 The length of the converter name in the resource specification was 0 or more than 8.

5 The length of the transaction name in the resource specification was 0 or more than 4.

6 The length of the CICS application program name in the resource specification was 0 or more than 8.

The response and reason codes are displayed in message DFHWB0723. An error response with a 400 (Bad Request) status code is returned to the Web client. This can be modified with the user-replaceable Web error program DFHWBEP.

URP_INVALID
The eye-catcher was invalid. This indicates an internal error.

Chapter 11. Converter programs

Converter programs are primarily for use with application programs that were not originally coded for use with the Web. They can also be used to combine output from several application programs into a single HTTP message.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Converter programs are not used when CICS is an HTTP client, or for Web service processing; they can only be invoked when CICS is an HTTP server. The role of converter programs in the CICS Web support process for CICS as an HTTP server is described in "HTTP request and response processing for CICS as an HTTP server" on page 26. Chapter 5, "Planning your CICS Web support architecture for CICS as an HTTP server," on page 59 has information to help you plan your architecture for CICS as an HTTP server.

A URIMAP definition can invoke a converter program to carry out relevant processing for HTTP requests. If an analyzer program is used in CICS Web Support processing, the analyzer program can also invoke a converter program. A converter program can be useful in the following circumstances:

- When application programs that were not originally coded for use with the Web need to receive input in the form of a COMMAREA, or need their output to be converted into an HTTP response. 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. You can use a converter program to perform this conversion or other processing on the content of the request.
- When you want to make more than one application program work on the same request data in sequence, and return a single HTTP response to the Web client.

If a converter program is invoked directly from a URIMAP definition, the PROGRAM attribute of the URIMAP definition (which specifies the name of the application program to process the request) can be passed to the converter program, and the converter program can choose to override it.

A converter program receives the Web client's request in a block of storage, together with a parameter list giving more information about the request. The converter program processes the content of the request into a format which is suitable for the application program that will provide data for the response, and passes it to the application program in a COMMAREA. This sequence is called the **decode** function of the converter program. If a converter program does not use the **decode** function to create a COMMAREA for the application program, the 32 767 byte buffer used to receive the HTTP request is passed to the application program.

The application program returns its results to the converter program. The converter program can invoke a further application program or programs, if more than one application program is needed to produce the data for the response. When the converter program has all the required data from the application programs, it

produces an HTTP response to be sent to the Web client. This sequence is called the **encode** function of the converter program.

A converter program is not associated with a TCPIPSERVICE definition in the same way as an analyzer program. You can use any converter program to process any HTTP request, but it must be local to the CICS system in which the request is received. For a given request, the same converter program is called for both the decode and encode functions.

All the user-replaceable programs must be local to the system in which CICS Web support is operating. If you do not use autoinstall for programs, you must define and install program definitions for all user-replaceable programs used by CICS Web support, including the analyzer and converter programs. If you use autoinstall for programs, you must ensure that user-replaceable programs are installed with the correct attributes.

Converter programs are also used by the CICS business logic interface. The role of the converter program in the CICS business logic interface is described in "Using the CICS business logic interface to call a program" on page 362. The caller of the CICS business logic interface determines whether a converter program is required, and which converter program should be called.

Writing a converter program

To write a converter program, you need to construct decode and encode functions, and consider code page conversion.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

You can write a converter program in Assembler, C, COBOL, or PL/I. Language-dependent header files, include files, and copy books are described in Appendix F, "Reference information for converter programs," on page 403.

Decode function: viewing and processing the HTTP request

The decode function of the converter program receives the HTTP request from the Web client, together with a parameter list giving more information about the request. The HTTP request is passed to the converter program in a 32K block of storage, which is indicated by a pointer in the parameter list. The request has already been divided into separate elements, such as the method, request headers and body. (Note that if the request is too long to fit into the block of storage, the remainder of the data is not passed to the converter program.) If an analyzer program is used in the processing path, the analyzer program might have modified the content of the request.

In a converter program for CICS Web support, you can use **EXEC CICS** WEB API commands to examine the HTTP request, if you prefer. The WEB EXTRACT command retrieves information about the request (such as the method and version). The WEB READ HTTPHEADER command or the HTTPHEADER browsing commands can be used to read the HTTP headers. The WEB RECEIVE command can be used to receive the body of the request. If you use any of the **EXEC CICS** WEB API commands, note that these commands return the original information from the Web client's request, and you cannot use them to see any

modifications that an analyzer program has made. Changes by an analyzer program are only visible in the parameter list and block of storage passed directly to the converter program.

The name of the user-written application program that should provide data for the response is supplied in the parameter list, either taken from the URIMAP definition for the request, or set by the analyzer program. If an analyzer program is used, it can provide additional information directly to the converter program in a user token.

Using the information which you have obtained about the Web client's request, the decode function of the converter program needs to:

- Determine whether processing should continue for the request, or whether CICS should return an error response to the Web client.
- Specify the name of the user-written application program that is to process the request and provide the response. If the name has already been input from a URIMAP definition or by an analyzer program, the converter program can accept or change this.
- Construct the COMMAREA that is passed to the user-written application program. The COMMAREA includes data from the Web client's request which has been converted into an acceptable input format for the application program. The block of storage containing the HTTP request can be reused, or a new COMMAREA can be specified.

Encode function: producing the response

When the user-written application program has carried out its processing using the input supplied by the converter program, the encode function of the converter program receives an output COMMAREA from the application program. Using this data, the encode function of the converter program needs to:

- Invoke further application programs, if more than one application program is needed to supply data. To do this, the encode function sets the loop response to call the decode function again. The decode function changes the name of the application program, and supplies appropriate input in a COMMAREA. The output is returned to the encode function again. “Calling more than one application program from a converter program” on page 145 has more information about this.
- Construct an HTTP response to be sent to the Web client.

In a converter program for CICS Web support, you can use **EXEC CICS** WEB API commands to produce and send the response to the Web client. The WEB WRITE HTTPHEADER command can be used to write HTTP headers for the response. The WEB SEND command can be used to assemble and send the response.

Alternatively, the converter program can construct the HTTP response manually in a buffer of storage, and return this to CICS for sending to the Web client. The response must contain an HTTP version, status code, status text, any HTTP headers that are required, and the message body. The format of the response should be compliant with the HTTP protocol specification to which you are working (HTTP/1.0 or HTTP/1.1). To obtain a buffer of storage for the HTTP response, you can:

- Issue a GETMAIN command to obtain storage.
- Use storage acquired in an earlier stage of processing (such as the analyzer program).

- Construct the response in the COMMAREA returned by the user-written application program.

The first word of the area used for the response must contain the length of the area (that is, the length of the HTTP response plus 4). On exit from the encode function of the converter program, the data pointer in the parameter list must point to this block of storage. (If you use **EXEC CICS** WEB API commands to send the response instead, CICS ignores and discards any block of storage indicated by this pointer.)

Whichever method you use to construct the HTTP response, CICS normally inserts some HTTP headers suitable for an HTTP/1.0 or HTTP/1.1 response, which are listed in Appendix B, "HTTP header reference for CICS Web support," on page 375. If the response produced by the converter program already contains these headers, CICS does not replace them. If the response has been flagged by an analyzer program for pre-CICS TS Version 3 compatibility processing, because a Web client requires a response identical with the response it would have received before CICS TS Version 3, only the headers that were sent to clients before CICS Transaction Server for z/OS, Version 3 Release 1 are used. This flag only works if the converter program produces the response manually in a block of storage. If the converter program uses the **EXEC CICS** WEB API commands to send the response, the flag has no effect.

Code page conversion

When you use **EXEC CICS** WEB API commands in a converter program to view the HTTP request and produce the response, code page conversion takes place as you specify in the commands, in the same way as for any other program which uses the **EXEC CICS** WEB API commands.

A converter program is not able to specify code page conversion settings for the HTTP request passed to it in the 32K block of storage. If a converter program is invoked directly from 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 block of storage 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, either use an analyzer program in the processing path to specify alternative code page conversion settings, or use the **EXEC CICS** WEB API commands to handle the request.

If your converter program constructs the HTTP response manually in a buffer of storage, CICS mirrors the code page conversion that was carried out for the request passed in the 32K block of storage. The response is sent to the Web client using the character set and host code page settings specified by the analyzer program, or in the absence of an analyzer program, the standard settings described in this topic. If the analyzer program suppressed code page conversion for the request, no code page conversion is carried out for the response body. If this outcome is not suitable, use the **EXEC CICS** WEB API commands to produce the response instead.

Converter programs for the CICS business logic interface

When you use a converter program with the CICS business logic interface, there are restrictions which might affect how you construct the COMMAREA that is passed to the user-written application program, and the buffer of storage containing the response. For more information, see "Offset mode and pointer mode" on page 369.

Do not use **EXEC CICS** WEB API commands in a converter program which is written for the CICS business logic interface.

Input parameters for converter program decode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Input parameters are passed to the decode function in a parameter list.

The parameters include:

- The IP address of the Web client.
- A pointer to the HTTP version of the Web client's request.
- A pointer to the request method.
- A pointer to the path component of the URL.
- A pointer to the HTTP headers for the request.
- A pointer to the entity body of the request message.
- The name of the CICS application program that provides data for the request (as set by the analyzer program, or specified in the URIMAP definition).
- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs" on page 133.
- An iteration counter which records the number of times the decode function has been entered for each HTTP request. The counter is set to 1 before the decode function is called for the first time, and is incremented before it is called on each subsequent occasion.
- An indication of whether the address of the entity body can be the target of a FREEMAIN command.

The analyzer program can change the values of any of these parameters before passing the parameter list to the converter program. If you want to examine the original request from the Web client, use the **EXEC CICS** WEB API commands in the converter program.

Output parameters for converter program decode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The decode function must provide the following outputs in a COMMAREA: a response code, and the address and length of the COMMAREA.

- A response code (optionally qualified by a reason code).

  If the decode function returns a response code of URP_OK, processing continues with the next step.

If the decode function returns any other value, the HTTP request is rejected with an error response. For details of the response made by CICS in this situation, see "CICS Web support default status codes and error responses" on page 122.

- The address and length of the COMMAREA passed to the user-written application program. If no application program is called, the COMMAREA is passed unchanged to the encode function.

The decode function may also provide the following outputs:

- The name of the user-written application program that is to provide data for the request. If the analyzer program supplied a name, the converter program can change it, or specify that no application program should be called.
- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs" on page 133.

Input parameters for converter program encode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Input parameters are passed to the encode function in a COMMAREA.

The parameters include:

- The address and length of the COMMAREA returned by the user-written application program. If no application program was called, the COMMAREA is passed unchanged from the decode function.
- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs" on page 133.
- An iteration counter that records the number of times the encode function has been entered for each HTTP request. The counter is set to 1 before the encode function is called for the first time, and is incremented before it is called on each subsequent occasion.

Output parameters for converter program encode function

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The encode function can provide the following outputs: a response code, the address of the storage buffer, the length of the HTTP response and an eight byte user token.

- A response code (optionally qualified by a reason):
  - If the encode function returns a response code of URP_OK, CICS sends the supplied HTTP response to the Web client, unless you have already used the **EXEC CICS** WEB API commands to do this.
  - If the encode function returns a response code of URP_OK_LOOP, processing continues with the decode function. See "Calling more than one application program from a converter program" on page 145 for more information.
  - If the encode function returns any other value, the HTTP request is rejected with an error response. For details of the response made by CICS in this situation, see "CICS Web support default status codes and error responses" on page 122.

- If you have constructed the HTTP response manually in a buffer of storage, the address of the buffer of storage, and the length of the HTTP response. The first word of the buffer must contain the length of the data (that is, the length of the HTTP response plus 4). If you use **EXEC CICS** WEB API commands to send the response instead, CICS ignores and discards any block of storage indicated by this pointer.
- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs" on page 133.

Calling more than one application program from a converter program

Sometimes, the data you need to construct the response to an HTTP request comes from more than one user-written application program.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

When this is the case, you can repeat the following sequence as necessary:

- The converter's decode function.
- An application program.
- The converter's encode function.

Do this by setting the response to URP_OK_LOOP in the encode function. When the HTTP response is complete, set the response to URP_OK.

When the decode function is called on the second and subsequent occasions, the following input parameters are not available:

- The HTTP version.
- The method.
- The path component of the URL.
- The request headers.
- The entity body.

However, you can use the WEB EXTRACT command to retrieve the same information.

Use the data pointer in the parameter list and the user token to share data between the decode and encode functions. When the encode function is called for the last time, if you are constructing the HTTP response manually in a buffer of storage, make sure that the data pointer (**encode_data_ptr**) addresses a valid HTTP response. If you are using **EXEC CICS** WEB API commands to produce and send the response, do so at this stage; in this situation, CICS ignores and discards any block of storage indicated by this pointer.

Chapter 12. HTTP client requests from a CICS application

CICS can act as an HTTP client, and communicate with an HTTP server on the Internet. A user-written application program sends requests through CICS to the HTTP server, and receives the responses from it.

In CICS Transaction Server for z/OS, Version 3 Release 1, the facility for CICS to act as an HTTP client is fully integrated into CICS Web support. You could use this facility in your user-written application programs to enable the applications to:

- Interact with hardware or software using the HTTP protocol (for example, printers can often 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.

Note that the HTTP client facility of CICS Web support is not designed for use as a browser. User application programs can make requests for individual, known resources that are available from a server, but they would not be expected to browse the Internet generally. The range of responses that you might receive from a server, and the actions that you need to take to handle them, should relate only to your preselected resources, plus the error responses that might be associated with those resources and with the type of requests you are making.

"HTTP request and response processing for CICS as an HTTP client" on page 30 explains the processing structure for CICS as an HTTP client. Before writing an application program that makes an HTTP client request, make sure you understand the processing stages for these requests, because most of the stages are initiated by the application program itself.

Several CICS Web support facilities are used when CICS is an HTTP client:

- You can use **EXEC CICS** WEB API commands in CICS application programs to open an HTTP connection to a server, make requests, and receive responses for processing by the application program. "Making HTTP requests through CICS as an HTTP client" on page 148 explains how to do this.
- Code page conversion is carried out for the requests CICS makes and the responses it receives. "Code page conversion for CICS as an HTTP client" on page 42 explains this process.
- URIMAP definitions may be used to avoid specifying information such as a URL or a certificate label in your application program. "Creating a URIMAP definition for an HTTP request by CICS as an HTTP client" on page 163 tells you how to create these.
- The global user exits XWBAUTH, XWBOPEN and XWBSNDO enable you to provide basic authentication credentials, specify the use of proxy servers, and to apply a security policy, for HTTP client requests. See "HTTP client send exit XWBAUTH" on page 165, "HTTP client open exit XWBOPEN" on page 168 and "HTTP client send exit XWBSNDO" on page 170 for more information about these exits.

TCPIPSERVICE resource definitions, which are used for CICS as an HTTP server, do not apply to CICS as an HTTP client, and you do not need to create these to make HTTP client requests.

Making HTTP requests through CICS as an HTTP client

HTTP client requests made from CICS to a server on the Internet are initiated by a user-written application program. This topic tells you how to write an application program that makes an HTTP client request.

Before you begin

Before writing an application program that makes an HTTP client request, read about the processing stages for these requests, because most of the stages are initiated by the application program itself. "HTTP request and response processing for CICS as an HTTP client" on page 30 explains what the application program needs to do, and what actions CICS takes during the process.

About this task

For CICS as an HTTP client, the application program makes requests to a server, and waits for the responses. An application program can control more than one connection, using a session token to differentiate between them.

To make HTTP requests and receive responses, write your application program to follow this process:

Procedure

1. Initiate a connection to the server, using the WEB OPEN command. "Opening a connection to an HTTP server" on page 149 tells you how to do this. This step ensures that the server is available, and generates the session token that can be used to manage the connection.
2. Write HTTP headers for the request, using the WEB WRITE HTTPHEADER command. "Writing HTTP headers for a request" on page 150 tells you how to do this. CICS automatically provides the headers that are required for HTTP/1.1 messages. You can provide additional headers for other purposes.
3. If required, produce an entity body, or message body, which is the content of the HTTP request. The process is the same as when CICS is an HTTP server, as described in "Producing an entity body for an HTTP message" on page 86. For many request methods, an entity body is not used, but for the POST and PUT methods (which are used to supply data to the server) it is required. The entity body can be formed from a CICS document (which is created using the **EXEC CICS** DOCUMENT application programming interface) or from a buffer of data supplied by the application program.
4. Send the request to the Web client using the WEB SEND or WEB CONVERSE command. "Writing an HTTP request" on page 152 tells you how to do this. You need to select a suitable method, and specify the entity body. CICS assembles the request using these items and the HTTP headers. If you want to send the entity body as series of small sections (or chunks), you also need to follow the special instructions in "Using chunked transfer-coding to send an HTTP request or response" on page 89.
5. If you want to send further requests as a pipelined sequence, use the guidance in "Sending a pipelined sequence of requests" on page 154 to do this.
6. If the server requests a username and password, use the guidance in "Providing credentials for basic authentication" on page 155 to provide these details.
7. Wait for and receive the request, following the process in "Receiving an HTTP response" on page 156:

a. Receive the message body of the response using the WEB RECEIVE command (or the WEB CONVERSE command that you issued earlier).
b. Read the headers for the response using the WEB READ HTTPHEADER command, or the HTTP header browsing commands.
c. Process the server's response, depending on the status code, and execute the application's business logic.
d. If you sent a pipelined sequence of requests, receive the rest of the responses from the server using further WEB RECEIVE commands.

8. If further requests and responses are wanted, repeat the process. If the server supports persistent connections, and does not close the connection, there is no need to open a new connection. You can continue using the same session token. If the server closes the connection, you need to issue the WEB OPEN command again if you want to make further requests.
9. When you have finished working with the server, close the connection. "Closing the connection to an HTTP server" on page 157 explains how to do this.

Opening a connection to an HTTP server

When making an HTTP client request in CICS Web support, you must open a connection to the server before sending the first request. CICS returns a session token that represents the connection.

Before you begin

About this task

Initiate a connection with the server by issuing a WEB OPEN command as follows:

Procedure

1. Specify the host name of the server, the length of the host name, and the scheme that is to be used (HTTP or HTTPS). Also specify the port number for the host if this is other than the default for the specified scheme. You can specify the URIMAP option on the WEB OPEN command to use this information directly from an existing URIMAP definition. Alternatively, you can supply the information using the SCHEME, HOST, HOSTLENGTH and PORTNUMBER options. You can extract these details from a known URL using the WEB PARSE URL command, or from an existing URIMAP definition using the WEB EXTRACT URIMAP command.
2. If required, specify the CODEPAGE option to change the EBCDIC code page for this connection to something other than the default code page for the local CICS region (set by the LOCALCCSID system initialization parameter). This might be the EBCDIC code page for another national language. When the server returns its response, if conversion is specified, CICS converts the response body into this code page before passing it to the application.
3. If you are using the HTTPS scheme, specify appropriate security options:
   a. If you need to supply an SSL client certificate, specify the CERTIFICATE option to do this. If you specify the URIMAP option on the WEB OPEN command, you can use this information directly from an existing URIMAP definition.
   b. Use the CIPHERS and NUMCIPHERS options to specify a list of cipher suite codes to be used for the connection. If you specify the URIMAP option

on the WEB OPEN command, you can either accept the setting from the URIMAP definition, or specify your own list of cipher suite codes to override the URIMAP specification.

4. If your first planned request involves actions which are not supported in all versions of the HTTP protocol, and you want to check the HTTP version of the server to confirm that the actions will succeed, specify either or both of the HTTPVNUM and HTTPRNUM options to return this information. You might need to do this if you do not already know the HTTP version of the server, and you want to take actions dependent on the HTTP protocol version, such as:
   - Writing HTTP headers that request an action which might not be carried out correctly by a server below HTTP/1.1 level.
   - Using HTTP methods that might be unsuitable for servers below HTTP/1.1 level.
   - Using chunked transfer-coding.
   - Sending a pipelined sequence of requests.

   Bear in mind that you do not always need to check the HTTP version of the server before carrying out actions dependent on the version. Consult the HTTP specification to which you are working to see whether it is acceptable to attempt the action with a server of the wrong version. For example, some unsuitable HTTP headers might be ignored by the recipient. You might be able to attempt the request without checking, and handle any error response from the server. Do not specify the HTTPVNUM and HTTPRNUM options if you do not require this information, because performance is better without these options.
5. The WEB OPEN command drives the XWBOPEN user exit. You can create a user exit program to make the connection to the server go through a proxy server, or to apply a security policy to a host name, if required. "HTTP client open exit XWBOPEN" on page 168 has information to help you do this.

Results

CICS opens the connection with the server, and returns a session token to the application program.

What to do next

Save the session token and use it on all subsequent commands that relate to this connection.

Writing HTTP headers for a request

For client HTTP requests, CICS automatically provides the HTTP headers that are required for basic messages, depending on the HTTP protocol version used for the message. You might need to add further HTTP headers to your request.

About this task

Some HTTP headers are created automatically by CICS if the message requires them. The full list of headers created by CICS is:

- ARM correlator
- Connection
- Content-Type (written by CICS, but can be supplied by a client application if a complex header is required)

- Content-Length
- Date
- Expect
- Host
- Server
- TE (written by CICS but further instances may be added)
- Transfer-Encoding
- User-Agent
- WWW-Authenticate

Some of these headers are appropriate only for CICS as an HTTP server. The circumstances in which these headers are created are described in Appendix B, "HTTP header reference for CICS Web support," on page 375. CICS does not allow you to write your own versions of CICS-supplied request headers, except for the Content-Type and TE headers.

The headers that CICS provides for a request are the ones that should normally be written for a basic HTTP/1.1 message to be compliant with the HTTP/1.1 specification. (CICS sends your request with HTTP/1.1 given as the HTTP version.) You might want to add further HTTP headers for purposes such as:

- Stating preferences to the server (for example, Accept-Encoding, Accept-Language)
- Making a conditional request (for example, If-Match, If-Modified-Since)
- Providing basic authentication information to a server or proxy (Authorization, Proxy-Authorization)

Check the HTTP specification to which you are working for requirements relating to any additional HTTP headers that you decide to use for your message. See "The HTTP protocol" on page 12 for more information about the HTTP specifications.

Write additional HTTP headers for a message **before** you issue the WEB SEND command to send the message. The exception to this rule is if you are writing headers to be sent as trailing headers on a chunked message, in which case the special process mentioned below applies. When writing HTTP headers for a request:

Procedure

- For all commands, specify the session token for this connection, using the SESSTOKEN option.
- Use the WEB WRITE HTTPHEADER command for each header that you want to add to the message. Make sure that you specify the name and value for each header in the format described by the HTTP specification to which you are working. The command adds a single header, and you can repeat the command to add further headers. If you write a header that you have already written for the request, CICS adds the new header to the request in addition to the existing header. Make sure that you only do this where the HTTP specification states that the header may be repeated. CICS stores the headers ready to be added to the request when it is sent.
- If you do not know the HTTP version of the server, and you want to use a header to request an action that might not be carried out correctly by a server below HTTP/1.1 level, use the WEB EXTRACT command to check the HTTP version of the server. Bear in mind that you do not always need to check the HTTP version of the server before carrying out actions dependent on the

version. Consult the HTTP specification to which you are working to see whether it is acceptable to attempt the action with a server of the wrong version. For example, some unsuitable HTTP headers might be ignored by the recipient. You might be able to attempt the request without checking, and handle any error response from the server.

- If you want to produce a date and time stamp for use in one of your HTTP headers (for example, the If-Modified-Since header), you can use the FORMATTIME command. The STRINGFORMAT option on the command converts the current date and time (in ABSTIME format), or a date and time produced by the application program, into suitable date and time stamp formats for use on the Web. Other date and time stamp formats might not be accepted by some Web clients or servers with which CICS is communicating.
- If you are using chunked transfer-coding to send an HTTP request, and you want to include trailing headers at the end of the chunked message, follow the special instructions in "Using chunked transfer-coding to send an HTTP request or response" on page 89. You need to write a Trailer header before sending the first chunk of the message. All the HTTP headers written after the WEB SEND command for the first chunk are treated as trailing headers.
- Make sure that your application program carries out any actions that are implied by your user-written headers.

Writing an HTTP request

For CICS as an HTTP client, the WEB SEND command or the WEB CONVERSE command can be used to make a request. The WEB CONVERSE command combines the options available on the WEB SEND command and the WEB RECEIVE command, so that you can use a single command to issue a request and receive the response.

Before you begin

You need to specify an HTTP method when making a request. The method tells the server what to do with your request. Appendix D, "HTTP method reference for CICS Web support," on page 391 provides basic guidance on the different methods that you can use with CICS Web support. For more detailed guidance, including any requirements that apply to your use of methods, you should consult the HTTP specification to which you are working. See "The HTTP protocol" on page 12 for more information about the HTTP specifications. CICS sends your request with HTTP/1.1 given as the HTTP version.

Write any additional HTTP headers for the request using the WEB WRITE HTTPHEADER command before making the request, as described in "Writing HTTP headers for a request" on page 150.

About this task

If wanted, the request can be sent in chunks (chunked transfer-coding), or you can send a pipelined sequence of requests.

The *CICS Application Programming Reference* has full reference information and descriptions of the options available on the WEB SEND and WEB CONVERSE commands. When you issue your chosen command:

Procedure

1. Specify the session token for this connection, using the SESSTOKEN option.

2. Specify the HTTP method for the request (OPTIONS, GET, HEAD, POST, PUT, DELETE, or TRACE). Appendix D, "HTTP method reference for CICS Web support," on page 391 has guidance for the correct use of each of these methods.
3. Specify the path information for the resource on the server that the application needs to access. The default is the path given in any URIMAP definition that you referenced on the WEB OPEN command for this connection. You can specify an alternative path by using the URIMAP option to name another URIMAP definition from which the path can be taken. (The new URIMAP definition must specify the correct host name for the current connection.) Alternatively, you can use the PATH and PATHLENGTH options to provide the path information.
4. Specify any query string for your request, using the QUERYSTRING and QUERYSTRLEN options.
5. Specify any entity body for the HTTP request, and its length. Appendix D, "HTTP method reference for CICS Web support," on page 391 has information about where the use of a request body is and is not appropriate.
   - For the GET, HEAD, DELETE, and TRACE methods, a request body is inappropriate.
   - For the OPTIONS method, a request body is permitted, but the HTTP/1.1 specification does not define any purpose for this body at present.
   - For the POST and PUT methods, a request body must be used.

   If a request body is required, the body content can be formed from a CICS document (using the CICS DOCUMENT interface and specifying the DOCTOKEN option to identify the document), or from the contents of a buffer (specifying the FROM option). "Producing an entity body for an HTTP message" on page 86 explains how to produce this, and has information about size limits for the length of the body.
6. Specify the media type for any entity body you are providing, using the MEDIATYPE option. For requests with the POST and PUT methods, which require a body, you need to specify the MEDIATYPE option. For requests with other methods, where no body content is provided, the MEDIATYPE option is not required.
7. If code page conversion is **not** required for the request body, specify the appropriate conversion option (depending on whether you are using the WEB SEND command or the WEB CONVERSE command) so that CICS does not convert the request body. For CICS as an HTTP client, the default setting is that the request body is converted, unless it has a non-text media type.
8. If code page conversion is required, and the default ISO-8859-1 character set is not suitable, specify a character set that is suitable for the server. ISO-8859-1 should be acceptable for most servers, unless you know from earlier connections that the server prefers an alternative.
9. If you want to use the Expect header to test the server's acceptance of the request, specify EXPECT for the ACTION option. This setting 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. The Expect header is not supported by servers below HTTP/1.1. If CICS does not yet know the HTTP version of the server, CICS makes an additional request before sending your request, in order to determine the HTTP version of the server. If the Expect header would not be suitable, CICS sends your request without it.

10. If this is the last request that you want to make to this server, specify CLOSE for the CLOSESTATUS option. CICS writes a Connection: close header on the request, or, for a server at HTTP/1.0 level, omits the Connection: Keep-Alive header. Specifying this option when you make your final request is good behavior, because 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 response from the server is still received and made available to your application. However, you will not be able to send any further requests to the server using this connection.
11. If you want to use chunked transfer-coding to send the request body as a series of chunks, follow the additional instructions in "Using chunked transfer-coding to send an HTTP request or response" on page 89.

    Note: Chunked transfer-coding is not supported with:
    - Servers below HTTP/1.1.
    - The WEB CONVERSE command.
    - CICS documents (the DOCTOKEN option).
12. If you want to send a pipelined sequence of requests, follow the instructions in "Sending a pipelined sequence of requests."

Results

CICS assembles the request line, HTTP headers and request body, and sends the request to the server.

Sending a pipelined sequence of requests

You may send further requests without waiting for a response from the server. This is known as pipelining. The WEB SEND command is used for sending pipelined requests, and the WEB CONVERSE command cannot be used (because that command includes waiting for a response).

Before you begin

About this task

"How CICS Web support handles pipelining" on page 37 has more information about pipelining. If you want to pipeline requests:

Procedure

1. Make sure you have a persistent connection with the server. The HTTP/1.1 specification allows you to make one attempt to send a pipelined sequence without checking that the connection is persistent. If this attempt fails, you must check before retrying the requests. To determine the nature of the connection:
    a. If you specified the HTTPVNUM and HTTPRNUM options on the WEB OPEN command for the connection, examine the returned information to determine the HTTP version of the server.
    b. If you did not specify those options on the WEB OPEN command, use the WEB EXTRACT command to determine the HTTP version of the server.
    c. If you have received a previous response from the server, use the WEB READ HTTPHEADER command to check if the server sent a Connection: close or a Connection: Keep-Alive header.

Servers that are at HTTP/1.1 level and **do not** send a Connection: close header, and servers that are at HTTP/1.0 level and **do** send a Connection: Keep-Alive header, support persistent connections. CICS does not carry out this check on your behalf, because CICS is not able to determine whether a client application program is sending a pipelined sequence of requests, as a pipelined request has no special headers to identify it.

2. The HTTP/1.1 specification says that your sequence of requests should be idempotent; that is, if you repeat all or part of the sequence, the same results should be obtained. "Pipelining" on page 19 has more information about idempotency.
3. Do not specify CLOSESTATUS(CLOSE) on any of the requests except the *final* request in the pipelined sequence.

Providing credentials for basic authentication

When an HTTP 401 WWW-Authenticate message is received, your application must provide the username and password (credentials) required by the server for basic authentication. Your application can also provide these credentials without waiting for the 401 message.

Procedure

1. Open a web session with the server using the WEB OPEN command, using the SESSTOKEN option. The SESSTOKEN will be returned to you when the session is opened successfully, and the session token must be used on all CICS WEB commands that relate to this connection.
2. Issue a WEB SEND command, specifying the SESSTOKEN for this connection. This WEB SEND command retrieves the realm from the server.
3. Issue a WEB RECEIVE command. The server returns a status code. Use the STATUSCODE option on the WEB RECEIVE command to check for a 401 response.
4. If the status code is 401 (the server requires authentication details), repeat your first WEB SEND request, but this time add the AUTHENTICATE(BASICAUTH) option. The XWBAUTH global user exit is called by the client application. This second WEB SEND command uses the realm received from the first WEB SEND command and the XWBAUTH exit to determine the required username and password.
5. You might prefer to specify AUTHENTICATE(BASICAUTH) in your initial WEB SEND command, instead of waiting for the 401 response. You can either:
   a. Supply your username and password in the WEB SEND command using the AUTHENTICATE(BASICAUTH) option.
   b. Invoke the XWBAUTH global user exit by specifying the AUTHENTICATE(BASICAUTH) option, but omitting your credentials. The user exit will be invoked, but the realm passed to the exit will be empty, as the realm has not yet been received from the server. The user exit must derive the required credentials from other parameters, for example, HOST and PATH.
6. If your application needs to know the realm that was sent in the 401 response, use the WEB EXTRACT command.

Results

CICS passes the username and password credentials to the server in an Authentication header.

Receiving an HTTP response

The WEB RECEIVE command or the WEB CONVERSE command is used to receive the response from the server. (The WEB CONVERSE command combines the functions of the WEB SEND and WEB RECEIVE commands.) The WEB READ HTTPHEADER command or the HTTP header browsing commands are used to examine the headers.

Before you begin

The time that the application is prepared to wait to receive a response is indicated by the RTIMOUT value specified on the transaction profile definition for the alias transaction. The timeout limit does not apply to reading the headers of the response.

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.

About this task

Using the WEB RECEIVE or WEB CONVERSE command:

Procedure

1. Specify the session token for this connection, using the SESSTOKEN option.
2. Specify data areas to receive the HTTP status code sent by the server, and any text returned by the server to describe the status code. Note that the data is returned in its unescaped form.
3. Specify a data area to receive the media type of the response body.
4. Receive the response body by specifying either the INTO option (for a data buffer), or the **SET** option (for a pointer reference), and the LENGTH option. The data is returned in its escaped form, and converted into a code page suitable for the application, unless you request otherwise.
5. If you want to limit the amount of data received from the response body, specify the MAXLENGTH option. If you want to retain, rather than discarding, any data that exceeds this length, specify the NOTRUNCATE option as well. Any remaining data can be obtained using further WEB RECEIVE commands. 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 entity body for the chunked message, rather than to each individual chunk.
6. If code page conversion is **not** required for the response body, specify the appropriate conversion option (depending on whether you are using the WEB RECEIVE command or the WEB CONVERSE command), so that CICS does not convert the response body. The default is that conversion should take place, and in that case CICS converts the body of the server's response into the default code page for the local CICS region, or any alternative EBCDIC code page that you specified on the WEB OPEN command.

   Note: If you receive an entity body that has been zipped or compressed, as indicated by a Content-Encoding header on the message, make sure that you suppress code page conversion. CICS does not decode these types of message for you, and if code page conversion is applied the results could be

unpredictable. If you do not want to receive zipped or compressed entity bodies, you can specify this using an Accept-Encoding header on your request to the server.

When you issue the WEB RECEIVE or WEB CONVERSE command, CICS returns the response body and the information from the status line.

7. Examine the HTTP headers of the server's response using the appropriate CICS commands:
   - If you want to read a specific HTTP header that you know the server provides, use the WEB READ HTTPHEADER command to examine the contents of that header. Your application program must provide a buffer which will receive the contents of the header. CICS returns a NOTFND condition if the header is not present in the request.
   - If you want to browse all the HTTP headers in the response, use a WEB STARTBROWSE HTTPHEADER command to begin browsing the header lines. Use a WEB READNEXT HTTPHEADER command to retrieve the header name and header value for each line. Your application program must provide two buffers: one will receive the name of the header, and one will receive its contents. CICS returns an ENDFILE condition when all headers have been read. Use a WEB ENDBROWSE HTTPHEADER command when your program has retrieved all the header information of interest.

   Remember to include the session token on each of the HTTP header commands.
8. Process the server's response and execute the application's business logic. If the response had a "normal" or informational status code, such as 200 (OK), you can process the response as normal. (The status code is received when you issue the WEB RECEIVE command.) If the response had a status code indicating an error or requesting further action, you should carry out alternative processing to account for this. Appendix C, “HTTP status code reference for CICS Web support,” on page 381 has basic guidance on responding to status codes.
9. If you sent a pipelined sequence of requests, receive the rest of the responses from the server using further WEB RECEIVE commands. CICS holds the responses and returns them to the application program in the order that CICS received them from the server. A server that handles pipelined requests will provide the responses in the same sequence in which the requests were received.

   Tip: When you are receiving responses to pipelined requests, if you are using multiple WEB RECEIVE commands to receive overlength message bodies, be careful to keep track of how many WEB RECEIVE commands you have issued. You might find it more convenient to receive the whole body for each of these responses in a single WEB RECEIVE command.

Closing the connection to an HTTP server

When CICS is an HTTP client, the connection between CICS and the server can be terminated by the server, or by CICS following a command issued by the application program, or at end of task.

Before you begin

About this task

To manage connection closure effectively, the following behavior is recommended:

Procedure

1. On the last request that you want to make to the server, specify CLOSE for the CLOSESTATUS option on the WEB SEND or WEB CONVERSE command. CICS writes a Connection: close header on the request, or, for a server at HTTP/1.0 level, omits the Connection: Keep-Alive header. Specifying this option when you make your final request is good behavior, because 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 response from the server is still received and made available to your application. However, you will not be able to send any further requests to the server using this connection. Specifying the CLOSESTATUS option does not have the same range of effects as issuing the WEB CLOSE command.
2. 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 terminates the connection, the application program cannot send any further requests using that connection, but it can receive responses that the server sent before it terminated the connection.
3. When all the HTTP requests and responses are completed, issue a WEB CLOSE command, specifying the session token. 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 you have completed all interaction with the server and with the last response that it sent. If the application program does not issue a WEB CLOSE command, the connection is closed at end of task.

Sample programs: pipelining requests to an HTTP server

Sample programs DFH$WBPA (Assembler), DFH$WBPC (C), and DFH0WBPO (COBOL) demonstrate how CICS can pipeline client requests to an HTTP server.

Before you begin

The sample programs send requests to a CICS region in which CICS Web support is running. The pipelined requests are handled by the CICS-supplied sample program DFH$WB1C. Before you use the sample programs, set up a CICS region as an HTTP server, following the procedure described in Chapter 4, “Configuring CICS Web support base components,” on page 53. Complete the procedure by setting up the sample program DFH$WB1C and the sample URIMAP definition DFH$URI1, as described in “Verifying the operation of CICS Web support” on page 56. Note that if your CICS region is already set up and operating as an HTTP server, and you have your own properly architected TCPIPSERVICE definitions, you must *not* install the sample TCPIPSERVICE definition HTTPNSSL again; just set up DFH$WB1C and DFH$URI1.

About this task

When you have set up a CICS region as an HTTP server, complete the following steps to use the pipelining sample programs:

Procedure

1. Identify the CICS region which will be the HTTP client. For the purpose of trying out the sample programs, you have three options:

- You can use the same CICS region as both the server and the client; the requests will go out of and into the region as they do with two separate CICS regions, and the results are the same. In this case, no further CICS Web support setup is required, because a CICS region that operates as an HTTP server can also operate as an HTTP client.
- You can use a different CICS region as the client, which has already been set up for CICS Web support. Again, in this case, no further CICS Web support setup is required.
- You can use a different CICS region as the client, which has not yet been set up for CICS Web support. In this case, you must carry out some basic CICS Web support setup, described in Step 2.

2. Optional: If you are using a different CICS region as the HTTP client, and the region has not yet been set up for CICS Web support, carry out basic setup as follows:
   a. Enable TCP/IP support for the CICS region, following the instructions in the *CICS Transaction Server for z/OS Installation Guide*. This process includes setting up Communications Server and establishing access to a DNS, or domain name, server through z/OS.
   b. Specify the system initialization parameter **TCPIP=YES** for the region to activate CICS TCP/IP services.

   This setup enables the CICS region to function as an HTTP client.
3. In the CICS region that you set up as an HTTP server, identify the TCPIPSERVICE definition for a port that the client region can use to make its requests. Select any port that is defined with the HTTP protocol, but does not use SSL; that is, with a TCPIPSERVICE definition that specifies PROTOCOL(HTTP) and SSL(NO). You can choose any suitable port because the sample URIMAP definition DFH$URI1, which is used on the server to access the sample program DFH$WB1C, matches any host name and port number.
4. In the HTTP client region, modify the supplied sample URIMAP definition DFH$URI2, which is provided in the DFH$WEB resource definition group. Because DFH$WEB is a protected group, copy the definition to another group to enable editing. DFH$URI2 is a URIMAP definition with a usage attribute of CLIENT. It specifies the components of the URL that the sample programs use to make the requests to the HTTP server region. Follow these steps to modify the sample:
   a. The scheme (SCHEME attribute) specified in DFH$URI2 is HTTP. Do not change this scheme.
   b. The host (HOST attribute) specified in DFH$URI2 is a dummy host name. Modify this name to insert the real host name as follows:
      - Specify the host name assigned to the z/OS image for the HTTP server region. If you do not know the host name, you can use the colon hexadecimal or dotted decimal IP address from the TCPIPSERVICE definition that you selected in Step 3.
      - If the TCPIPSERVICE definition that you selected is for a port number other than 80 (the well-known port number for HTTP), specify the port number from the TCPIPSERVICE definition after the host name, with a colon separating the host name and port number.
   c. The path (PATH attribute) specified in DFH$URI2 is `/sample_web_app`, which is matched by DFH$URI1. Do not change this path.
5. In the HTTP client region, install your modified URIMAP definition DFH$URI2.

6. In the HTTP client region, install the PROFILE definition DFH$WBPF, which is provided in the DFH$WEB resource definition group.
7. Translate and compile one of the sample programs in the required language. The pipelining sample programs are not compiled when you receive them. The sample programs are supplied in the SDFHSAMP library. The names of the sample programs and their corresponding transactions are as follows:

Language	Program	Transaction
Assembler	DFH$WBPA	WBPA
C	DFH$WBPC	WBPC
COBOL	DFH0WBPO	WBPO

8. In the HTTP client region, install the PROGRAM resource definition and the corresponding TRANSACTION resource definition for your chosen sample program. The resource definitions are provided in the DFH$WEB resource definition group.
9. In the HTTP client region, run the transaction for your chosen sample program. The sample program sends information messages to your terminal when it sends each of the three HTTP requests successfully, receives each of the three HTTP responses successfully, and completes. If any of the sends or receives fail, an error message is given instead. The content of the HTTP requests and responses is not displayed.
10. When you have finished using the sample program, for security reasons, disable the URIMAP definitions DFH$URI1 and DFH$URI2, and uninstall the sample TCPIPSERVICE definition HTTPNSSL if you were using it.

Related concepts:

"Pipelining" on page 19
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.

"How CICS Web support handles pipelining" on page 37
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.

Related tasks:

"Verifying the operation of CICS Web support" on page 56
Sample programs DFH$WB1A (Assembler) and DFH$WB1C (C) help you to test that CICS Web support is working. The sample programs use EXEC CICS WEB and DOCUMENT commands to receive your request and construct and send a simple response.

Sample programs: Sending and receiving HTTP requests in chunks

Sample programs DFH$WBCA (Assembler), DFH$WBCC (C), and DFH0WBCO (COBOL) demonstrate how CICS, as an HTTP client, can send a request in sections or chunks to an HTTP server, and receive a chunked message in response. Sample programs DFH$WBHA (Assembler), DFH$WBHC (C), and DFH0WBHO (COBOL) demonstrate how CICS, as an HTTP server, can receive a request in chunks from an HTTP client and send a chunked response.

Before you begin

The sample programs send and receive requests between CICS regions in which CICS Web support is running. The client chunking samples, DFH$WBCA, DFH$WBCC, and DFH0WBCO, are installed in the HTTP client region, and the server chunking samples, DFH$WBHA, DFH$WBHC, and DFH0WBHO, are installed in the HTTP server region. The client sample, for example, DFH$WBCA, opens a session with its corresponding server sample, DFH$WBHA. DFH$WBHA receives the chunked request from DFH$WBCA and sends a chunked response. The client sample, DFH$WBCA, receives the response as a chunked message.

Before you use the sample programs, you need to set up a CICS region as an HTTP server, following the procedure described in Chapter 4, "Configuring CICS Web support base components," on page 53. If your CICS region is already set up and operating as an HTTP server, and you have your own properly architected TCPIPSERVICE definitions, you should not install the sample TCPIPSERVICE definition HTTPNSSL again.

About this task

When you have set up a CICS region as an HTTP server, complete the following steps to use the chunking sample programs:

Procedure

1. Identify the CICS region that will be the HTTP client. For the purpose of trying out the sample programs, you have three options:
   - You can use the same CICS region as both the server and the client; the requests will go out of and into the region as they would with two separate CICS regions, and the results will be the same. In this case, no further CICS Web support setup is required, because a CICS region that operates as an HTTP server can also operate as an HTTP client.
   - You can use a different CICS region as the client, which has already been set up for CICS Web support. Again, in this case, no further CICS Web support setup is required.
   - You can use a different CICS region as the client, which has not yet been set up for CICS Web support. In this case, you need to carry out some basic CICS Web support setup, described in Step 2.
2. Optional: If you are using a different CICS region as the HTTP client, and the region has not yet been set up for CICS Web support, carry out basic setup as follows:
   a. Enable TCP/IP support for the CICS region, following the instructions in the *CICS Transaction Server for z/OS Installation Guide*. This process includes setting up Communications Server and establishing access to a DNS, or domain name, server through z/OS.
   b. Specify the system initialization parameter `TCPIP=YES` for the region to activate CICS TCP/IP services.

   This setup enables the CICS region to function as an HTTP client.
3. In the CICS region which you set up as an HTTP server, identify the TCPIPSERVICE definition for a port that the client region can use to make its requests. Select any port that is defined with the HTTP protocol, but does not use SSL, so with a TCPIPSERVICE definition that specifies PROTOCOL(HTTP) and SSL(NO). You can choose any suitable port because the sample URIMAP

definition DFH$URI4, which is used on the server to access the server chunking sample program, matches any host name and port number.

4. In the HTTP client region, modify the supplied sample URIMAP definition DFH$URI3, which is provided in the DFH$WEB resource definition group. As DFH$WEB is a protected group, you need to copy the definition to another group to enable editing. DFH$URI3 is a URIMAP definition with a usage attribute of CLIENT. It specifies the components of the URL that the sample programs use to make the requests to the HTTP server region.
   a. The scheme (SCHEME attribute) specified in DFH$URI3 is HTTP. You do not need to change this.
   b. DFH$URI3 specifies a dummy host name (HOST attribute). Modify this to insert the real host name as follows:
      - Specify the host name assigned to the z/OS image for the HTTP server region. If you do not know the host name, you can use the IP address from the TCPIPSERVICE definition that you selected in Step 3.
      - If the TCPIPSERVICE definition that you selected is for a port number other than 80 (the commonly used port number for HTTP), you need to specify the port number from the TCPIPSERVICE definition after the host name, with a colon separating the host name and port number.
   c. The path (PATH attribute) specified in DFH$URI3 is `/chunking_sample_application`, which is matched by DFH$URI4. You do not need to change this.
5. In the HTTP client region, install your modified URIMAP definition DFH$URI3.
6. In the HTTP client region, install the PROFILE definition DFH$WBPF, which is provided in the DFH$WEB resource definition group.
7. Translate and compile a client and a server sample program in the language that you want to use. The chunking sample programs are not compiled when you receive them. The sample programs are supplied in the SDFHSAMP library. The names of the sample programs and their corresponding transactions are:

Type of Sample	Language	Program	Transaction
Client chunking sample	Assembler	DFH$WBCA	WBCA
Client chunking sample	C	DFH$WBCC	WBCC
Client chunking sample	COBOL	DFH0WBCO	WBCO
Server chunking sample	Assembler	DFH$WBHA	-
Server chunking sample	C	DFH$WBHC	-
Server chunking sample	COBOL	DFH0WBHO	-

8. In the HTTP server region, install the PROGRAM resource definition for your chosen server chunking sample program, and install the supplied sample URIMAP definition DFH$URI4. The resource definitions are provided in the DFH$WEB resource definition group.
   a. If you chose the C or COBOL sample rather than the Assembler sample, you need to modify the supplied sample URIMAP definition DFH$URI4

before installing it. As DFH$WEB is a protected group, you need to copy the definition to another group to enable editing.

b. Change the program (PROGRAM attribute) specified by DFH$URI4, from DFH$WBHA (the Assembler chunking sample program), to your preferred server chunking sample program.

c. Install your modified URIMAP definition DFH$URI4.

9. In the HTTP client region, install the PROGRAM resource definition and the corresponding TRANSACTION resource definition for your chosen client chunking sample program. The resource definitions are provided in the DFH$WEB resource definition group.
10. In the HTTP client region, run the transaction for your chosen client chunking sample. The sample program outputs information messages to your terminal when it sends all four chunks of the message and two header trailers to the HTTP server successfully. A message is also displayed confirming that the receive occurred. If any of the sends fail, an error message is given instead. The content of the actual HTTP requests and responses is not displayed.
11. In the HTTP server region, your chosen server chunking sample is called by the corresponding client chunking sample. The sample program outputs information messages to your terminal when it sends all four chunks of the message and two header trailers to the waiting HTTP client successfully. If any of the sends fail, an error message is given instead. The content of the actual HTTP requests and responses is not displayed.
12. When you have finished using the sample programs, for security, you should disable the URIMAP definitions DFH$URI3 and DFH$URI4, and uninstall the sample TCPIPSERVICE definition HTTPNSSL, if you were using it.

Creating a URIMAP definition for an HTTP request by CICS as an HTTP client

You can create a URIMAP definition which specifies the components of the URI for an HTTP client request (scheme, host and path), and an SSL client certificate to be used with the request, if required.

About this task

You can name a URIMAP definition on the WEB OPEN command, to provide a scheme and host name and a default path for the connection. You can also name it on a WEB SEND command, to provide a path for the relevant request. Alternatively, you can use the WEB EXTRACT URIMAP command to extract information from the URIMAP definition and use it directly in the application program that makes the HTTP client request. URIMAP definitions can also be used with the INVOKE SERVICE command to call a service.

Procedure

1. Identify the URL that you plan to use for the HTTP client request . The URL represents a resource that you plan to access on a server.
2. Identify whether a client certificate might be required for the request, and obtain a suitable certificate label. If the scheme used for the request is HTTPS (HTTP with SSL), the server might request a SSL client certificate. If this happens, CICS supplies the certificate label that is specified in the URIMAP definition.
3. Divide the URL for the request into its scheme, host and path components. "The components of a URL" on page 10 explains each of these components

and how they are delimited. You should also use a port number if it has been specified explicitly in the URL. For example, in the URL `http://www.example.com:1030/software/index.html`:

- The scheme component is `http`
- The host component is `www.example.com`
- The port number is `1030`
- The path component is `/software/index.html`

If you want to provide a query string in the URL for the request, you can specify this on the WEB SEND command using the QUERY option.

4. Begin a URIMAP definition with a name and group of your choice.
5. Use the STATUS attribute to specify whether the URIMAP definition should be installed in an enabled or disabled state.
6. Specify a USAGE attribute of CLIENT (CICS as an HTTP client).
7. Specify the SCHEME attribute as the scheme component of the URL for the request. HTTP (without SSL) or HTTPS (with SSL) can be used. Do not include the delimiters `://` following the scheme component.
8. Specify the HOST attribute as the host component of the URL for the request. The host component can be an explicit IPv4 or IPv6 address, or a character host name. 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).
9. Specify the PATH attribute as the path component of the URL for the request.
   a. Do not include a query string in the path component; you can specify it on the WEB SEND command using the QUERY option.
   b. Do not use a wildcard character (an asterisk) in a URIMAP definition for CICS as an HTTP client.
   c. You can either include or omit the forward slash at the beginning of the path component. If you omit it, CICS adds it at runtime.

   If the URIMAP definition is referenced on a WEB OPEN command, this path becomes the default path for WEB SEND commands for 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.
10. Optional: If SSL is being used, specify the CERTIFICATE attribute as the label of the certificate that is to be used as the SSL client certificate for this request.
11. Optional: If SSL or TLS is being used, specify the CIPHERS attribute as the cipher code that is to be used for this request.

Example

This example shows the URL `http://www.example.com:1030/software/index.html` specified as a URIMAP definition:

```
Urimap       ==> softw
Group        ==> MYGROUP
Description  ==> Client request for software page
STatus       ==> Enabled
USAge        ==> Client
```

```
UNIVERSAL RESOURCE IDENTIFIER
    SCheme       ==> HTTP
    HOST         ==> www.example.com:1030
    PAth         ==> /software/index.html
```

HTTP client send exit XWBAUTH

With XWBAUTH, you can specify basic authentication credentials (username and password) for a target server or service provider. XWBAUTH passes these to CICS on request, to create an Authorization header which is forwarded using HTTP. XWBAUTH is called during processing of an **EXEC CICS** WEB SEND (Client) or **EXEC CICS** WEB CONVERSE command. The host name and path information are passed to the user exit, with an optional qualifying realm.

When AUTHENTICATE(BASICAUTH) is specified in the **EXEC CICS** WEB SEND (Client) or WEB CONVERSE command, the application can provide a username and password. If they are not supplied, XWBAUTH is called, providing an alternative way of specifying these credentials.

The username and password are usually specific to the remote server environment, and might be longer than the standard 8 characters used by RACF systems. The username and password fields can be up to 256 characters in length. The syntax of these fields is not validated.

The host is passed to the user exit program as the UEPHOST parameter, and the path is passed as the UEPPATH parameter. The realm is passed optionally as the UEPREALM parameter. In response, the user exit program returns the username and password as the UEPUSNM and UEPPSWD parameters. A return code of UERCNORM indicates a successfully returned username and password. Return code UERCBYP indicates that the username and password cannot be identified, so the Authorization header is not added to the request. A return code of UERCERR indicates that the exit cannot supply credentials, and that the requesting CICS command will fail.

The following sample exit programs are shipped in the CICS sample library, SDFHSAMP:

- DFH$WBPI
- DFH$WBEX
- DFH$WBX1
- DFH$WBX2
- DFH$WBGA, a copybook to map the global work area used by the DFH$WBPI, DFH$WBX1, DFH$WBX2, and DFH$WBEX samples.

For more information about the client sample exit programs, see *CICS Customization Guide*. For more information about setting up your LDAP profile, see *CICS RACF Security Guide*.

Exit XWBAUTH

When invoked

When the **EXEC CICS** WEB SEND or WEB CONVERSE command specifies AUTHENTICATE(BASICAUTH), but the USERNAME and PASSWORD are not specified.

Exit-specific parameters

UEPHOST (Input supplied by CICS)
The address of a field containing the address of the host name, IPv4, or IPv6 address specified in the HOST option of the WEB OPEN command for the connection. The host name is converted into lowercase characters when it is saved in this field. Your user exit program must take this conversion 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.

UEPPATH (Input supplied by CICS)
The address of a field containing the address of the path specified in the PATH option of the WEB SEND or WEB CONVERSE command. The path is mixed case, as it was specified.

UEPPATHL (Input supplied by CICS)
The address of a field containing the halfword length of the path.

UEPREALM (Input supplied by CICS)
The address of a field containing the address of the realm name associated with the target destination, if a realm name was returned in a previous HTTP 401 response from the server.

UEPREALML (Input supplied by CICS)
The address of a field containing the halfword length of the realm name.

UEPAUTHT (Input supplied by CICS)
The address of a 1-byte code that indicates the authentication type. This is a binary 01, indicating Basic Authentication.

UEPUSNM (Output supplied by user exit)
The address of a fullword field, containing the address of the username required to access the HTTP server. A predefined address and 64-byte area are created by CICS to store the username. You can place your username in this 64-byte area, leaving the address in UEPUSNM unchanged. Alternatively, you can place your username in your own area and replace the address in UEPUSNM with your username address. If you create your own username area, the field can be up to 256 bytes in length.

UEPUSNML (Input supplied by CICS and output supplied by user exit)
The address of a halfword field, which initially contains the length of the buffer address supplied in UEPUSNM. Your user exit program must set the length of this buffer to the actual username length, as supplied in UEPUSNM.

UEPPSWD (Output supplied by user exit)
The address of a fullword field, containing the address of the password required to access the HTTP server. A predefined address and 64-byte area are created by CICS to store the password. You can place your password in this 64-byte area, leaving the address in UEPPSWD unchanged. Alternatively, you can place your password in your own area and replace the address in UEPPSWD with the address of your password. If you create your own password area, the field can be up to 256 bytes in length.

UEPPSWDL (Input supplied by CICS and output supplied by user exit)
The address of a halfword field, which initially contains the length of

the buffer address supplied in UEPPSWD. Your user exit program must set the length of this buffer to the actual username length, as supplied in UEPPSWD.

UEPHOSTT (Input supplied by CICS)
The address of a 1-byte code that indicates the host type contained in the UEPHOST parameter.

Binary 01 indicates host name, binary 02 indicates an IPv4 address, and binary 03 indicates an IPv6 address.

Return codes

UERCNORM
The exit has successfully returned a username and password.

UERCBYP
The exit cannot identify a username and password. An Authorization header is not sent.

UERCERR
The exit cannot identify a username and password. The WEB SEND (Client) or WEB CONVERSE command must be stopped.

XPI calls
All XPI calls can be used.

API and SPI commands
All API and SPI commands can be used, except for **EXEC CICS SHUTDOWN** and **EXEC CICS XCTL**.

Typical use of the LDAP XPI functions by XWBAUTH

The expected use of the DFHDDAPX functions (in association with the XWBAUTH global user exit) include opening and closing an LDAP session, browsing results for credentials, scanning and locating results, closing the browse, returning the correct value and closing the search.

BIND_LDAP
Establishes a session with an LDAP server. Used once on the first call to the global user exit XWBAUTH. The LDAP session token is stored in XWBAUTH's global work area (if one is provided) for use by subsequent calls to LDAP_SEARCH.

UNBIND_LDAP
Releases the connection with the LDAP server. This function is only required during CICS shutdown processing. This function can be used during the XSTERM (system termination) global user exit.

SEARCH_LDAP
Searches for credentials, specifying an LDAP distinguished name, that identifies the URL and realm of the required user information. Distinguished name is specified in the following format:

```
racfcid=uuuuuuuu, ibm-httprealm=rrrrrrrr, labeledURI=xxxxxxxx, cn=BasicAuth
```

where:

- `uuuuuuuu` is the current userid, obtained from the XWBAUTH parameter, UEPUSER.
- `rrrrrrrr` is the HTTP 401 realm, obtained from the XWBAUTH parameter, UEPREALM (if this exists).

- `xxxxxxxx` is the target URL, obtained by concatenating `http://` with the hostname from the XWBAUTH parameter, UEPHOST, and the path from the XWBAUTH parameter, UEPPATH.
- `cn=BasicAuth` is an arbitrary suffix that is configured into the LDAP server for storing Basic Authentication credentials.

START_BROWSE_RESULTS
Starts scanning the results returned by SEARCH_LDAP.

GET_NEXT_ENTRY
Locates the next result entry in a series of entries returned by SEARCH_LDAP. Typically, the URL specified in SEARCH_LDAP will locate a unique entry and the GET_NEXT_ENTRY function is not used.

GET_NEXT_ATTRIBUTE
Locates the next attribute in the current result entry. Typically, specific attributes will be selected and the GET_NEXT_ATTRIBUTE function is not used.

END_BROWSE_RESULTS
Ends the browse session started by SEARCH_LDAP.

GET_ATTRIBUTE_VALUE
Returns the values for various attributes of the target distinguished name. For XWBAUTH, these attributes values are the username and password, stored in the attributes `uid` and `userpassword`. XWBAUTH returns these attribute values as credentials.

FREE_SEARCH_RESULTS
Closes the search initiated by SEARCH_LDAP and releases associated storage.

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. XWBOPEN is also called during processing of an **EXEC CICS INVOKE SERVICE** command.

CICS itself does not have any requirements concerning the use (or otherwise) of proxy servers for HTTP requests by CICS as an HTTP client, and CICS does not apply any security policy for those requests. It is your responsibility to set up these facilities if they are required by your system or organization.

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, a NOTAUTH 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** or **EXEC CICS INVOKE SERVICE** command.

Exit-specific parameters

UEPHOST (Input supplied by CICS)
The address of a field containing the host name, IPv4, or IPv6 address 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. The proxy server name must be in URL format. 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.

UEPHOSTT (Input supplied by CICS)
The address of a 1-byte code that indicates the host type contained in the UEPHOST parameter.

Note: Binary 01 indicates host name, binary 02 indicates an IPv4 address, and binary 03 indicates an IPv6 address.

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.

CICS itself does not apply any security policy for HTTP requests by CICS as an HTTP client; it is your responsibility to set up this facility if it is required by your system or organization.

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, a NOTAUTH 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 IPv4, or IPv6 address 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.

UEPHOSTT
The address of a 1-byte code that indicates the host type contained in the UEPHOST parameter.

Note: Binary 01 indicates host name, binary 02 indicates an IPv4 address, and binary 03 indicates an IPv6 address.

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.

Chapter 13. Security for CICS Web support

When CICS is connected to the Internet, security measures are essential to prevent unauthorized access to CICS applications and data, and also to prevent third parties obtaining private information that is sent over the Internet.

You should consider security throughout the development process for your CICS Web support architecture, as part of the design of your CICS Web support applications and utility programs, as well as when creating resource definitions for the relevant CICS facilities. This section summarizes the measures that can be used to enhance the security of your CICS Web support implementation.

Authentication and identification for HTTP clients

Authentication and identification of clients enables a server to protect its resources from access by unauthorized users.

CICS as an HTTP server: authentication and identification

For CICS as an HTTP server, authentication schemes are specified by the AUTHENTICATE attribute of the TCPIPSERVICE definition. Identification is obtained in connection with the authentication process, or can be supplied by CICS if authentication is not needed.

Obtaining authentication and identification from Web clients is a key step in protecting your CICS system from access by unauthorized users.

TCPIPSERVICE resource definitions are the main place where you specify the security measures that are applied for CICS as an HTTP server. You need a TCPIPSERVICE resource definition for each port that you use for CICS Web support. The TCPIPSERVICE resource definition specifies:

- Whether or not SSL is used for the port.
- The authentication scheme that is used for the port.
- The realm for basic authentication.

Authentication

Two authentication schemes are supported by CICS for use with the HTTP protocol:

- **Basic authentication** is an HTTP facility that enables a client to both authenticate and identify itself to a server by providing a user ID and password. This information is encoded using base-64 encoding, which is simple to decode. Because of this, using basic authentication as the sole means of authentication is only appropriate when there is no possibility of a password being intercepted. In most environments, basic authentication should be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information. “HTTP basic authentication” on page 20 explains basic authentication in more detail.
- **SSL client certificate authentication** is a more secure method of authenticating a client, using a client certificate which is issued by a trusted third party (or Certificate Authority), and sent using SSL encryption. *CICS RACF Security Guide* explains how this works. A client certificate does not contain a user ID that can

be used for identification within CICS. To achieve identification, the client certificate can be associated with a user ID in RACF or an equivalent security manager, either before the certificate is used, or automatically (using basic authentication) when the client makes its request. The RACF user ID becomes the client's user ID each time the certificate is used. *CICS RACF Security Guide* explains how to set this up.

"Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 tells you how to set up a TCPIPSERVICE definition for CICS Web support which specifies one of these authentication schemes.

When you use basic authentication or client certificate authentication, CICS handles the process of requesting authentication from the user, decoding the authentication information if necessary, checking the supplied authentication against the security manager's database, and rejecting the request if the authentication is not acceptable. An analyzer program or user-written application program is only called after the authentication has been verified and accepted.

All the user IDs used by Web clients must have a user profile in RACF , or your equivalent external security manager. *CICS RACF Security Guide* has more information about these.

For basic authentication, if the password supplied by the user is found to have expired, CICS prompts the user for a new password and helps them to re-submit their request. The CICS-supplied utility program DFHWBPW is used to do this. You can customize the text on the Web pages that CICS displays to the user during this process. "Password expiry management for HTTP basic authentication" on page 176 has the information you need to do this.

For client certificate authentication, CICS verifies the supplied certificate by checking it against the security manager's database, and (optionally) against any certificate revocation list that you have set up. A user-written application can examine information obtained by this process, if this is useful for determining how to process the request. Use the EXTRACT CERTIFICATE command to retrieve:

- Components of the issuer's or the subject's distinguished name. *CICS RACF Security Guide* explains distinguished names.
- The RACF user ID associated with the certificate.

Identification

Identification takes place when you obtain a user ID for the Web client. The ID can be obtained from the Web client:

- During basic authentication.
- By the association of a user ID with a client certificate.

For application-generated responses only, it is also possible for CICS to supply a user ID on behalf of the Web client:

- In an analyzer program that is used in the processing path for the application-generated response. (This can override a user ID obtained for the Web client.)
- In the URIMAP definition for the request. (This cannot override a user ID obtained for the Web client.)
- As the CICS default user ID, if no other can be determined.

It is important to note that if you supply a user ID on behalf of the Web client, there is no authentication of the client's identity. You should only do this when communicating with your own client system, which has already authenticated its users, and communicates with the server in a secure environment. *CICS RACF Security Guide* explains in more detail how the user ID is determined, depending on the settings for the TCPIPSERVICE definition.

When the client has been identified, the client's user ID can be authorized for access to CICS resources like any other user ID, using RACF or an equivalent external security manager. You can choose to apply resource level security to any or all of the individual resources which the Web client is accessing in CICS, such as Web pages stored as CICS document templates or z/OS UNIX files, or CICS commands used by the application which provides the response. "CICS system and resource security for CICS Web support" on page 178 explains how to secure these resources, and how to remove resource level security if you do not want it.

CICS as an HTTP client: authentication and identification

When you make an HTTP client request through CICS, a server or proxy might require you to perform basic authentication, proxy authentication, or SSL client certificate authentication. Basic authentication can be performed using the AUTHENTICATE option of your WEB SEND or WEB CONVERSE command. Proxy authentication is carried out by your user application. A client certificate can be supplied using a URIMAP definition.

Your client application might be asked to authenticate itself in the following ways:

- **Basic authentication** allows you to provide a username and password for access to specific information. When you make a request to a server, the server might send you a response with a 401 status code, and a WWW-Authenticate header. The header names the realm for which basic authentication is required. To receive the information you requested, you need to provide the username and password, and CICS re-sends the request with an Authorization header, specifying your credentials (the username and password) to allow you access to the realm. It is also possible for CICS to send an Authorization header directly to a server that is expecting it, which eliminates the need for a 401 response. CICS converts the username and password to ASCII and applies base-64 encoding, as required by the basic authentication protocol. This allows you to supply your credentials in normal characters through the WEB SEND or WEB CONVERSE command, or through the XWBAUTH user exit. "Providing credentials for basic authentication" on page 155 has information about the steps required to set up basic authentication within your application. "HTTP basic authentication" on page 20 has more information about basic authentication.
- **Proxy authentication** is initiated by a proxy server. For proxy authentication, the status code for the response is 407, the challenge header from the proxy server is Proxy-Authenticate, and the response header is Proxy-Authorization. CICS does not support this protocol.
- **SSL client certificate authentication** uses a client certificate which is issued by a trusted third party (or Certificate Authority). A server might or might not require you to provide this authentication when you are making an HTTPS request. The *CICS RACF Security Guide* tells you how to obtain a certificate and store it in a key ring in the RACF database, or equivalent external security manager. If a server does request a client certificate, CICS supplies the certificate label which is specified in the URIMAP definition that was used on the WEB OPEN command for the connection. Alternatively, the certificate label can be directly specified as an option in the WEB OPEN command. (If you use a

URIMAP definition but do not specify a certificate label, the default certificate defined in the key ring for the CICS region user ID is used.)

Some servers might ask you to provide other types of authentication or identification. If you are unable to provide acceptable authentication or identification to a server, your request will be rejected. For basic authentication or proxy authentication, the status code used when a server rejects your request is the same as the status code for the challenge (401 for a server or 407 for a proxy). If you respond to a challenge but then receive a further response with one of these status codes, the authorization information you used is not valid.

Password expiry management for HTTP basic authentication

When basic authentication is used for an HTTP connection, CICS Web support checks the user ID and password in the external security manager. If the password has expired, the CICS-supplied utility program DFHWBPW is used to prompt the user to select a new password. You can customize or replace the pages presented to the user by DFHWBPW.

DFHWBPW is used only for password expiry management when the TCPIPSERVICE definition that applies to the request is defined with the BASIC, AUTOREGISTER, or AUTOMATIC option for the AUTHENTICATE attribute. Although DFHWBPW has a structure similar to a converter program, it is not part of the normal CICS Web support processing path, so you do not need to add code to it for any other purpose. When the user has selected their new password, DFHWBPW restarts the request submission by redirecting the client to the URL for the original request, so that the complete processing path for the request occurs as normal.

DFHWBPW presents two Web pages to the user:

1. Password prompt page. This page contains two elements:
   a. A message about password validity. The initial message displayed to the user states that the password has expired. If there is a problem with the user's attempt to change the password (for example, the two supplied copies of the new password do not match), further messages are displayed to explain the problem.
   b. An HTML form for the user to change their password.
2. Confirmation and request refresh page. This page confirms that the expired password has been successfully replaced, and provides a refresh tag and URL link so that the request can be remade automatically or manually.

DFHWBPW builds these web pages using three CICS document templates, DFHWBPW1, DFHWBPW2, and DFHWBPW3. The CICS-supplied definitions for these templates define them as loadable programs: that is, they are of type PROGRAM(DFHWBPW1) and so on. The definitions are in the CICS-supplied RDO group DFHWEB. You can change these definitions by copying them to another group and using the RDO ALTER command to change them so that the templates are derived from a different source. Alternatively, you can leave the RDO definitions unchanged, and modify the programs that are loaded instead. The three programs DFHWBPW1, DFHWBPW2, and DFHWBPW3 are assembler language data-only modules, and their source is shipped to you in corresponding members of the CICS sample library, SDFHSAMP. You can modify these samples and reassemble and linkedit them into one of your normal CICS program libraries that are concatenated into the DFHRPL data definition statement.

Tip: When you code ampersands (&) in Assembler language you have to type them as double ampersands (&&).

The content and function of each of the DFHWBPW templates is as follows:

DFHWBPW1
Part of the password prompt page. Provides the HTML page heading for the page, and sets symbols for the possible password validity messages (using the server-side include technique for setting symbols). The messages convey the following information to the user:

message.1
Password has expired.

message.2
The entered userid is invalid.

message.3
The two copies of the proposed new password do not match.

message.4
The previous password entered (the one that has just expired) is not correct.

message.5
The proposed new password is not permitted by the external security manager, because of password quality rules.

message.6
The userid has now been revoked.

The DFHWBPW program selects the appropriate symbol to insert into the document for the password prompt page. You can customize DFHWBPW1 to change the page heading and title, or alter the `body` tag to change the page colors or background. You can also change the content of the message symbols.

DFHWBPW2
Part of the password prompt page. Builds an HTML form where the user can input a user ID, the old (expired) password, and two identical copies of a proposed new password. You can customize DFHWBPW2 to change the text used to prompt the user, or otherwise change the layout of the page. However, you must not modify the contents of the `form` tag, or any of the `input` tags. If you do, DFHWBPW may not work as intended.

DFHWBPW3
Confirmation and request refresh page. The text notifies the user that the expired password has been successfully replaced, and explains that the user will shortly be prompted by the client to enter the password again, and that the new password should then be re-entered. You can customize the text and layout of the page.

DFHWBPW3 is designed to restart the request process. It contains a `meta http-equiv="Refresh"` tag that causes an automatic redirection after ten seconds to the page that the user had originally requested when the expired password was detected. You can change the time limit on this tag or remove it altogether if you do not want users to be redirected automatically. However, the modified page should always contain a link forward to the originally requested page. The URL for that page is in the symbol `&dfhwbpw_target_url;` . Restarting the request process means that if

the Web client has cached the old password, this can be replaced with the new password right away, and also means that the CICS Web support processing path is unaffected.

CICS system and resource security for CICS Web support

When CICS is an HTTP server, the CICS system must be protected from access by unauthorized users. If a system is not properly protected, users might be able to access confidential data, or obstruct the system to cause denial of service to other users.

To police access to CICS Web support in general, you should request identification from each user that makes an HTTP client request, and authenticate the identity stated by the user. The TCPIPSERVICE definitions for inbound ports are used to specify these requirements."CICS as an HTTP server: authentication and identification" on page 173 explains how this can be achieved for CICS as an HTTP server.

All the user IDs used by Web clients must have a user profile in RACF , or your equivalent external security manager. RACF user profiles has more information about these.

When you have obtained an authenticated user ID for a Web client, you can use this ID to implement resource level security for the resources in the CICS region which you are using to provide the response. The procedure varies for each type of response which you are providing:

- Application-generated responses.
- Static responses, using a URIMAP definition that provides a CICS document template as the response.
- Static responses, using a URIMAP definition that provides a z/OS UNIX Systems Services file as the response.

For application-generated responses, CICS system defaults specify that no resource security checking is carried out, but transaction security checking is carried out (specifically, transaction-attach security for the alias transaction). Assuming that transaction security is active in your CICS region, you therefore need to take some actions relating specifically to security for application-generated responses, even if you do not plan to use Web clients' authenticated user IDs for security checking.

For static responses, transaction-attach security does not apply to Web clients' user IDs. However, CICS system defaults specify that resource level security checking **is** carried out if a user ID is available for Web clients. If you are obtaining authenticated user IDs from Web clients, you therefore need to either set up resource permissions for these user IDs, or take action to disable resource level security checking.

Whether or not you choose to implement resource level security using Web clients' user IDs for every response provided by CICS Web support, you still need to ensure that you:

- Implement measures to protect inbound ports against unauthorized or malicious access.
- Protect CICS system components from modification by unauthorized users, and ensure that authorized users have the correct access to them.

Related information:

Resource level security for static responses using HFS files

Implementing security for HFS files

Security for inbound ports

Each port used for CICS Web support is defined by a TCPIPSERVICE resource definition. The TCPIPSERVICE definition specifies security options for the port, including whether or not SSL is used, and the level of authentication that is requested from clients. Ports need to be guarded as much as possible against unauthorized or malicious access.

"Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 explains how to create TCPIPSERVICE definitions for ports used for CICS Web support.

To help keep ports secure, bear these points in mind:

- Specify the MAXDATALEN attribute on every TCPIPSERVICE definition. This option limits the maximum amount of data that CICS will accept for a single request, and it helps to defend CICS against denial of service attacks involving the transmission of large amounts of data.
- Use Secure Sockets Layer (SSL) wherever you want to ensure that your interaction with the Web client remains confidential and cannot be intercepted by a third party. The use of SSL is particularly important where confidential data is being transmitted, or where authorization such as a user ID and password is being passed to the server. "SSL with CICS Web support" on page 184 explains how SSL is used with CICS Web support.

If you do experience unusual activity on one or more of your CICS Web support ports, you can use CICS system commands to shut down CICS Web support at different levels (a single request, a virtual host, a port, or the whole of CICS Web support), without shutting down the CICS system. "Rejecting HTTP requests" on page 109 explains how to do this.

URIMAP resource definitions name either HTTP or HTTPS (HTTP with SSL) as the scheme for the request. A URIMAP specifying the HTTP scheme accepts Web client requests made using either the HTTP scheme, or the more secure HTTPS scheme. A URIMAP specifying the HTTPS scheme accepts only Web client requests made using the HTTPS scheme.

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.

Security for CICS system components

CICS system components cannot be accessed directly by a Web client in the course of a normal request, unless you have Web-enabled a CICS-supplied transaction that can issue CICS system commands. However, as with any other CICS resource, it is important to protect these components from modification by unauthorized users.

You also need to ensure that authorized users, particularly the CICS region, have the required authority to use these components.

A number of components such as application programs and resource definitions are used to control CICS Web support. These components are listed in "Components of CICS Web support" on page 22. If you do not secure these against unauthorized access, the security of your CICS Web support architecture might be compromised. For example, a user with access to the TCPIPSERVICE definition for a port could remove the requirement for a Web client to use SSL or to provide identification. *CICS RACF Security Guide* explains how to secure CICS transactions, resources and commands against unauthorized use.

For some CICS system components, you might need to set up additional authorities to allow access to authorized users:

- For URIMAP definitions, additional authority might be required to set a user ID for the Web client. If surrogate user checking is enabled in the CICS region (with XUSER=YES specified as a system initialization parameter), CICS checks that the user ID used to install the URIMAP definition, is authorized as a surrogate of the user ID specified for the USERID attribute.
- Document templates can be used to produce the body of a response from CICS as an HTTP server, or the body of a request from CICS as an HTTP client. They are defined by DOCTEMPLATE resource definitions. If the document templates are stored in partitioned data sets, the CICS region user ID must have READ authority for the data set.
- z/OS UNIX Systems Services files can be used to produce the body of a static response from CICS as an HTTP server. They can either be specified under their own names, or defined by DOCTEMPLATE resource definitions. When a z/OS UNIX file is used, the CICS region must have permissions to access z/OS UNIX, and it must have permission to access the z/OS UNIX directory containing the file, and the file itself. *Java Applications in CICS* explains how to grant these permissions.

Resource and transaction security for application-generated responses

If you have obtained an authenticated user ID for a Web client (which has a profile in your security manager), this user ID is applied to the alias transaction that is used for the application-generated response. You either need to give appropriate permissions to the Web clients' user IDs, or supply your own standard user ID as an override. Whether or not you decide to use Web clients' user IDs for resource security checking, you need to ensure the user ID for the alias transaction has the appropriate permissions.

Before you begin

About this task

Alias transactions are defined by TRANSACTION resource definitions. The alias transaction for each application-generated response is specified by the URIMAP definition for the request, or by an analyzer program. The default is the CICS-supplied alias transaction CWBA. This is the case when either Web-aware applications or COMMAREA applications are used to provide the response.

The user ID under which the alias transaction runs must have authority to:

- Attach the alias transaction, if transaction-attach security is specified for the CICS region. Transaction-attach security is controlled by the system initialization parameter XTRAN. The default for this is YES (transaction-attach security is active).
- Access any CICS resources used by the alias transaction, if resource security is specified for the alias transaction. Resource security is controlled by the RESSEC attribute in the TRANSACTION resource definition for the alias transaction. The default for this is NO (no resource security), and NO is also the supplied setting for CWBA.
- Access any CICS system programming commands used by the alias transaction, if command security is specified for the alias transaction. These system programming commands would be used in the user-written application program that produces the response. Command security is controlled by the CMDSEC attribute in the TRANSACTION resource definition for the alias transaction. The default for this is NO (no command security), and NO is also the supplied setting for CWBA.

When a Web client makes a request to CICS Web support, and the response is provided by an application, CICS selects a user ID for the alias transaction in the following order of priority:

1. A user ID that you set using an analyzer program. This user ID can override a user ID obtained from the Web client or supplied by a URIMAP definition.
2. A user ID that you obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client. If authentication is required for the connection but the client does not provide an authenticated user ID, the request is rejected.
3. A user ID that you specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

Depending on your CICS Web support architecture, you might be using one or several of these types of user ID, for different requests. If you obtain an authenticated user ID for a Web client, this is used for the alias transaction unless you take action to override it.

You need to take the following security actions for application-generated responses:

Procedure

1. If you are obtaining authenticated user IDs for Web clients, but you do **not** want to use these for security checking for your application-generated responses, you need to use an analyzer program to override Web clients' user IDs with a standard user ID for the relevant alias transactions. (You could use the CICS default user ID.) The analyzer program must be placed in the processing paths for the requests where you want to supply this override. Chapter 10, “Analyzer programs,” on page 125 explains how to write an analyzer program, and how to use it in conjunction with a URIMAP definition. Make sure that this user ID has a user profile defined in your security manager. When you have set up a standard user ID, you can give the required permissions, as described in the remaining steps of this procedure, to the standard user ID.
2. If you are not obtaining authenticated user IDs for Web clients, select suitable user IDs to be standard user IDs for your alias transactions. Unless you just want to use the CICS default user ID, specify your chosen user IDs in the

URIMAP definitions for the requests, or set up an analyzer program to specify them. Make sure that the standard user IDs have user profiles defined in your security manager.

3. Assuming that transaction-attach security is specified for the CICS region, you need to ensure that all the possible user IDs for your alias transactions have authority to attach the transaction. This could include Web clients' user IDs, if you are obtaining them and are not overriding them, or a standard user ID that you have specified in a URIMAP definition or analyzer program, or just the CICS default user ID.the *CICS RACF Security Guide* explains how to give transaction-attach permissions to users.
4. Optional: If you want to apply resource level security checking for the resources used by an alias transaction:
   a. Identify all the CICS resources used by the alias transaction, and determine which of them are subject to resource security checking in your CICS region. Resources that might be used by an application program for CICS Web support, and the system initialization parameters that control resource security checking for them, include:
      - CICS document templates (XDOC system initialization parameter).
      - Other application programs invoked by the main application program to perform business logic (XPPT system initialization parameter).
      - Temporary storage queues used to share application state across an HTTP request sequence (XTST system initialization parameter).
      - Files managed by CICS file control (XFCT system initialization parameter).

      Note: Resource security checking for HFS files (XHFS system initialization parameter) does not apply when HFS files are used by an application program. This is because the files can only be manipulated by an application program when they are defined as CICS document templates, and it is CICS document template security which controls access to them in this situation.

      the *CICS RACF Security Guide* explains how to set up resource security checking for any of these resources, if you have not already done this.

      If you are using an analyzer program, this is the main program for the alias transaction, and so is not subject to resource security checking (only to the transaction-attach security checking). However, note that the user-written Web application program itself, and any converter program that you use, will be subject to separate resource security checking. Similarly, if you are using a converter program but no analyzer program, the converter program is the main program for the alias transaction, but the application programs called by the converter program are subject to separate resource security checking.
   b. Give all the user IDs that are permitted to attach the alias transaction, permission to use the secured resources used by the alias transaction.
   c. Specify RESSEC(YES) in the TRANSACTION resource definition for the transaction.
5. Optional: If you want to apply command security checking for any CICS system programming commands used by an alias transaction:
   a. Confirm that command security is active in the CICS region. Command security is activated by the XCMD system initialization parameter.

b. Identify the CICS system programming commands used by the application program or programs, analyzer program (if used), and converter program (if used), that are associated with the transaction. the *CICS RACF Security Guide* has a checklist of commands.
c. Give all the user IDs that are permitted to attach the alias transaction, permission to use the commands used by the alias transaction.
d. Specify CMDSEC(YES) in the TRANSACTION resource definition for the alias transaction.

What to do next

For any security checking to take place in a CICS region, the system initialization parameter SEC=YES must be set.

Resource level security for static responses using document templates

For static responses delivered by CICS Web support using a CICS document template specified in a URIMAP definition, resource security checking is enabled by default. If you have implemented basic authentication or client certificate authentication, and you also want to control users' access to specific Web pages, you can use Web clients' authenticated user IDs to control access to individual CICS document templates which you are using to provide static responses.

Before you begin

About this task

Resource security for CICS document templates is controlled by the **XRES** system initialization parameter. The default for this parameter is YES, meaning that resource security is active. If you do **not** want to use resource security checking for CICS document templates used for any purpose in your CICS region, you can deactivate it by setting this system initialization parameter to NO.

The transaction for all static responses is the default Web listener transaction CWXN, or any alternate transaction that you have specified in place of CWXN using the TRANSACTION attribute on your TCPIPSERVICE definitions. For CICS document templates, resource security checking can also be controlled by the RESSEC attribute in the TRANSACTION resource definition for the transaction. For CWXN, as supplied by CICS, RESSEC(YES) is specified, meaning that resource security is active. If you do **not** want to use resource security checking for static responses, the best way to deactivate it is to replace CWXN in your TCPIPSERVICE definitions with an alternate transaction that specifies the program DFHWBXN, and has RESSEC(NO). This deactivates resource security checking for CICS document templates for static responses only. (Note that the RESSEC attribute cannot control security checking for z/OS UNIX files specified by the HFSFILE attribute.)

Note: Document templates can be retrieved from a variety of sources, including partitioned data sets, CICS programs, CICS files, z/OS UNIX System Services files, temporary storage queues, transient data queues, and exit programs. When resource security checking is carried out for a document template, CICS does **not** perform any additional security checking on the resource that supplies the document template, even if resource security is specified for that type of resource in the CICS region.

To set up resource level security for static responses using CICS document templates:

Procedure

1. Identify the authenticated user IDs used by Web clients. These must be the basis of your resource security checking. (You cannot supply an override using an analyzer program, as you can with application-generated responses.) Authenticated user IDs will already have a user profile defined in your security manager.
2. Identify all the CICS document templates that you are using to provide static responses.
3. Implement security for CICS document templates in your CICS region, following the instructions in Security for CICS document templates. You will need to define a profile to your security manager for each CICS document template that you are using to provide a static response, and give permissions to access appropriate CICS document templates to each authenticated user ID.
4. Ensure that `RESSEC(YES)` is specified in the TRANSACTION resource definition of CWXN, or the alternate transaction that you have specified in place of CWXN. RESSEC(YES) is specified in CWXN as supplied by CICS, but for TRANSACTION resource definitions in general, the default is RESSEC(NO). This step activates resource security checking for your static responses, so ensure that whenever a Web client supplies a user ID, the appropriate permissions have been set up.

What to do next

For any security checking to take place in a CICS region, the system initialization parameter SEC must be set to YES.

SSL with CICS Web support

The Secure Sockets Layer (SSL) can be used with HTTP to enable encryption, message authentication, and client and server authentication using certificates. The HTTPS scheme is HTTP with SSL. When you have configured CICS to use SSL, its facilities are available for both CICS as an HTTP server, and CICS as an HTTP client.

The *CICS RACF Security Guide* explains the facilities that SSL provides and tells you how to make SSL work with CICS.

When CICS is an HTTP server, you can use SSL to protect an interaction with a Web client. To do this, specify appropriate security options on the TCPIPSERVICE definition for the port on which CICS receives the client's requests.

As well as specifying the use of SSL, you can require basic authentication or require a client certificate. To give more assistance to Web clients, you can allow a client to provide a client certificate, and then register themselves to the security manager to supply identification for the CICS environment. You can also allow a client to use self-registration or basic authentication as needed to supply identification. All these activities are handled by CICS itself, so if you are providing an application-generated response, your application does not need to handle this. “Creating TCPIPSERVICE resource definitions for CICS Web support” on page 96 explains how to create TCPIPSERVICE definitions that include these security options.

When CICS is an HTTP client, a server might require the use of SSL for some connections. If that is the case, you need to do some or all of the following:

- Use HTTPS as the scheme for the connection.
- Supply a list of cipher suites that you want to use for the connection. You can specify these in the URIMAP definition that you use on the WEB OPEN command for the connection.
- Supply a client certificate. Client certificates are not a requirement for all SSL transactions, but a server might require one for particular transactions. If a server does request a client certificate, you can specify the label of a suitable certificate in the URIMAP definition that you use on the WEB OPEN command for the connection, or on the WEB OPEN command itself. The client certificate must be stored in your security manager's key ring. (If you use a URIMAP definition but do not specify a certificate label, the default certificate defined in the key ring for the CICS region user ID is used.)

Chapter 14. CICS Web support and non-HTTP requests

You can use CICS Web support to process inbound TCP/IP client requests which are not in the HTTP format. In CICS Transaction Server for z/OS, Version 4 Release 1, this facility is primarily intended to provide support for requests from user-written clients that use nonstandard request formats. The processing that takes place for requests, and the response that is provided, are defined by the user. No specific support is provided for any formally defined protocols which are used for client-server communication.

CICS Web support only handles non-HTTP messages when CICS is the server. Non-HTTP requests cannot be made by CICS as a client. Client requests made through CICS Web support use the HTTP protocol.

When CICS Web support facilities are used for handling non-HTTP requests:

- You can use TCPIPSERVICE resource definitions to control the ports on which requests are received.
- You can use an analyzer program to assemble and parse requests, specify code page conversion, and determine subsequent request processing. You can code the analyzer program to parse requests in accordance with any request format that you have defined, but note that this CICS facility does not provide specific support for any particular protocol for which a formal definition exists.
- You can use either Web-aware application programs, or non-Web-aware applications with a converter program, to provide responses to requests. Requests and responses can be handled using certain elements of the EXEC CICS WEB programming interface, or passed between CICS applications in a COMMAREA.
- The Web error program DFHWBEP provides an error response if an abend occurs in the analyzer program, converter program, or user-written application program, and also if the analyzer program and converter program cannot determine what application program should be executed to service the request. The standard HTTP error messages are used by default, but you can tailor these if required.

Some CICS Web support facilities are not available for non-HTTP requests:

- Some of the facilities that help you interpret HTTP requests and construct the responses are not available. For example, message headers cannot be accessed separately.
- The enhancements introduced in CICS TS Version 3, including chunked transfer-coding, are generally not available to non-HTTP requests.
- Persistent connections are not supported.
- URIMAP definitions are not used for non-HTTP requests.

The support that CICS Web support provides for non-HTTP messages is not the same thing as the TCP/IP Sockets interface for CICS. The z/OS Communications Server IP CICS Sockets interface provides an application programming interface to allow clients to communicate directly with CICS application programs over TCP/IP. CICS Web support is not involved with this process.

The CICS Sockets interface is supplied with z/OS Communications Server, not with CICS. *z/OS Communications Server: IP CICS Sockets Guide*, SC31-8807, describes the CICS Sockets interface.

Handling non-HTTP requests

To handle non-HTTP requests using CICS Web support facilities, you need to code an analyzer program to determine processing for the requests, and application programs to provide responses to the requests. You also need to create some resource definitions.

Before you begin

The base components of CICS Web support must be configured before you start, as described in Chapter 4, "Configuring CICS Web support base components," on page 53.

About this task

The following components of CICS Web support are used for processing non-HTTP requests:

- TCPIPSERVICE resource definitions.
- An analyzer program.
- Converter programs, if required.
- User-written application programs.
- An alias transaction for the application programs.
- The Web error program DFHWBEP.

URIMAP definitions are not used with non-HTTP requests.

Processing for HTTP requests and processing for non-HTTP requests are kept separate. Non-HTTP requests are received using the USER protocol, specified on the TCPIPSERVICE definition. This ensures that CICS can perform basic acceptance checks on HTTP requests and responses, and that non-HTTP requests are not subjected to these checks. The acceptance checks would produce an error response for non-HTTP requests and the request would not be processed.

To use CICS Web support to handle non-HTTP requests:

Procedure

1. Decide on the port that will be used for the requests. Note that because only one active TCPIPSERVICE definition can exist for each port, non-HTTP requests cannot 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 making non-HTTP requests must explicitly specify the port number in the URL for their requests.
2. Set up resource definitions for the requests, using the information in "Resource definition for non-HTTP requests" on page 189. The TCPIPSERVICE definitions for non-HTTP requests must specify the USER protocol. You can also create alias transactions to cover request processing.
3. Code an analyzer program to handle each request, using the information in "Analyzer programs and non-HTTP requests" on page 189. The analyzer program is required to determine processing for the request. It specifies the application program and (if used) converter program to handle each request. It can also specify code page conversion parameters.
4. Design and code one or more application programs to provide a response to each request, using the information in "Application programming for non-HTTP requests" on page 191. The programs can use certain elements of the

EXEC CICS WEB programming interface. They may also be non-Web-aware applications, and produce output that is encoded by a converter program.

5. Ensure that the Web error program DFHWBEP provides appropriate responses in error situations. For non-HTTP requests, DFHWBEP is used if an abend occurs in the analyzer program, converter program, or user-written application program, and also if the analyzer program and converter program cannot determine what application program should be executed to service the request. By default, DFHWBEP outputs the standard HTTP messages that would be sent as error responses for HTTP requests in the same situations, but you can tailor these if required. Chapter 9, “Web error program,” on page 115 explains the situations in which DFHWBEP is used, and how to tailor the messages it provides.

Resource definition for non-HTTP requests

TCPIPSERVICE and TRANSACTION resource definitions are needed for non-HTTP requests. TCPIPSERVICE resource definitions for non-HTTP requests must specify the USER (user-defined) protocol, which is associated with the CICS-supplied transaction CWXU. URIMAP resource definitions are not used when requests are received through the USER protocol.

Before you begin

About this task

Procedure

1. Create a TCPIPSERVICE resource definition, with the USER protocol, for each port that you use for non-HTTP requests. The attributes that can be used with the USER protocol are the same as those which can be used with the HTTP protocol. “Creating TCPIPSERVICE resource definitions for CICS Web support” on page 96 tells you how to do this.
2. For each TCPIPSERVICE resource definition, decide whether to use the CICS-supplied transaction CWXU, the CICS Web user-defined protocol attach transaction, or an alternative. The DFHCURDI sample includes a sample definition for CWXU. CWXU executes the CICS program DFHWBXN. 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.
3. Optional: Create TRANSACTION resource definitions for any alias transactions that you want to use for request processing. “Creating TRANSACTION resource definitions for CICS Web support” on page 99 tells you how to do this.

Analyzer programs and non-HTTP requests

An analyzer program is required for processing non-HTTP requests. It can reconstruct requests that have been divided up for transmission across the network, specify code page conversion of the requests, and perform any parsing that is required to determine subsequent request processing.

Reconstructing a non-HTTP request

An incoming request may be divided into several parts for transmission across the network. For non-HTTP requests, CICS does not reconstruct the request before calling the analyzer program, and you should write your analyzer code accordingly.

On entry to the analyzer, the user_data pointer addresses a COMMAREA which contains the first part of the incoming request. To receive the next part of the request, set the return code to URP_EXCEPTION and the reason code to URP_RECEIVE_OUTSTANDING. CICS Web support invokes the analyzer again, and the user_data pointer addresses the next part of the message. You can repeat this process as many times as you need to until the entire request has been received, up to the maximum supported length of 32767 bytes.

The results of this process are not visible to the CICS WEB API commands. However, the reconstructed message can be passed to a converter program.

Specifying code page conversion for non-HTTP requests

For non-HTTP requests, CICS Web support does not perform any code page conversion on a request before the analyzer program is invoked.

The analyzer can specify code page conversion of non-HTTP requests as it can for HTTP requests, using either a code page conversion table (DFHCNV) key, or the client and server code page output parameters. “Writing an analyzer program” on page 128 explains how to do this.

Alternatively, a Web-aware application program can specify code page conversion of incoming non-HTTP requests on a WEB RECEIVE command.

Note that non-HTTP requests are not parsed into the request line, header and body elements. Any code page conversion that is carried out, must be carried out for the whole of the request.

Determining non-HTTP request processing

The following input fields which relate to HTTP requests are undefined in an analyzer program for non-HTTP requests:

- The HTTP version
- The method
- The path component of the request
- The request headers

The subsequent processing stages must therefore be determined by examining the content of the request.

The analyzer program can specify subsequent request processing by a converter program, or by a Web-aware application program. “Writing an analyzer program” on page 128 explains the inputs and outputs from an analyzer program, and how these are used to determine request processing.

Application programming for non-HTTP requests

Application programs for non-HTTP requests can use certain elements of the **EXEC CICS WEB** programming interface. They may also be non-Web-aware applications, and produce output that is encoded by a converter program.

A pseudoconversational programming model is not suitable for non-HTTP requests. Applications should be designed to receive a single request and provide a single response.

Web-aware applications

If you want to use a Web-aware application to respond to non-HTTP requests, you can use the following CICS API commands:

- The WEB RECEIVE command can be used to receive a non-HTTP request. If the application is used to respond to both HTTP and non-HTTP requests, you can use the TYPE option on the WEB RECEIVE command to distinguish between the two request types. Note that CICS does not carry out any parsing for a non-HTTP message, and requests that have been divided up for transmission across the network are not automatically assembled. If an analyzer program assembles the request, the results of this are not visible to the CICS WEB API commands.
- The **EXEC CICS DOCUMENT** commands can be used to compose a CICS document to form the body of a response.
- The WEB SEND command can be used to send a response to a non-HTTP client. However, the following options that relate to HTTP-specific actions are not suitable:
  - STATUSCODE and STATUSTEXT. If specified, these are ignored.
  - CLOSESTATUS. If specified, this is ignored.
  - CHUNKING. If specified, this causes an error on the command.
- The WEB RETRIEVE command can be used to retrieve a CICS document sent in an earlier **EXEC CICS** WEB SEND command.

Other **EXEC CICS** WEB commands relate to HTTP requests only, and can result in an INVREQ condition if used with non-HTTP requests.

An application program can specify code page conversion of non-HTTP requests using the **WEB RECEIVE** command.

Non-Web-aware applications with converter programs

With non-Web-aware applications, you can use a converter program to convert the input from the Web client into a suitable COMMAREA for the application, and to convert the output from the application into HTML to provide the response. If an analyzer program has reconstructed the request after it was divided up for transmission across the network, the results of this can be passed to a converter program.

The following input fields which relate to HTTP requests are undefined in a converter program for non-HTTP requests:

- The HTTP version
- The method
- The path component of the URL
- The request headers

Chapter 11, “Converter programs,” on page 139 has more information about writing converter programs.

Chapter 15. CICS Web support and 3270 display applications

When a 3270 transaction is accessed by a Web client, CICS can display the output as an HTML form. Use the variants of the Web Terminal Translation Application (DFHWBTTA, DFHWBTTB or DFHWBTTC) to provide Web clients with access to applications that were originally designed to use the 3270 display system.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

An HTML form can be created from the output of a 3270 transaction in one of two ways:

- For applications that use BMS, an HTML template is generated from a BMS map, and stored in the template library. You can customize the generation of the template. However, if the only changes you need to make to the generated HTML can be accommodated in the heading or footing section, you do not need to generate a template from the BMS map, as the map can be processed at execution time to generate the HTML form.
- For applications that do not use BMS, the outbound 3270 data stream is processed at execution time to generate the HTML form.

The Web Terminal Translation Application can be used to display the HTML forms to a Web browser.

Note: The Web Terminal Translation Application operates at HTTP/1.0 level. It does not make full use of the facilities available in CICS Web support (such as the **EXEC CICS** WEB API), and so does not provide compliance with the HTTP/1.1 specification. This means that:

- Requests from the Web client, and responses from the application, are not checked against the HTTP protocol specification.
- CICS does not provide HTTP/1.1 responses, in normal or error situations, even if the client is at HTTP/1.1 level.

All three variants of the Web Terminal Translation Application support non-conversational, conversational, and pseudoconversational transactions.

- DFHWBTTA and DFHWBTTB perform the translation between 3270 data streams and HTML, and between templates generated from BMS maps and HTML. Use DFHWBTTA if your HTML templates are 32767 bytes (32K) of data or smaller, and DFHWBTTB if your HTML templates are larger than 32K. (Using DFHWBTTB for smaller HTML templates incurs an unnecessary performance overhead.)
- DFHWBTTC performs the translation between BMS maps and HTML when no template is generated. BMS maps used in this way must specify TERM=3270 or omit the TERM parameter. DFHWBTTC supports HTML output of any length. Use DFHWBTTC if you do not need to generate HTML templates.

DFHWBTTB and DFHWBTTC are aliases for DFHWBTTA; the same program (DFHWBTTA) is invoked in each case. CICS uses the name by which the program is invoked to determine which processing is needed.

DFHWBTTA, DFHWBTTB and DFHWBTTC generate HTML that conforms to the HTML 3.2 specification. If you use a Web browser that does not support HTML 3.2, some functions may not work correctly.

HTML generated for terminals having a pagesize which results in a field position greater than 4095 (x'FFF') might not function correctly, particularly when using DFHWBTTC. The exception to this is when using old style templates. (Old style templates are those generated by DFHWBTLG from CICS TS 1.2 or CICS TS 1.3 before PTF UQ53534). Code has been supplied to tolerate BMS sends of such templates when using DFHWBTTA or DFHWBTTB, but not DFHWBTTC.

You can create URIMAP definitions that specify DFHWBTTA, DFHWBTTB or DFHWBTTC as the program to be invoked to process a request (PROGRAM attribute). The method that the Web client uses to access the program is similar, but the use of URIMAP definitions gives you an online administration facility that can be used to prevent or redirect requests. When a URIMAP definition is used, the use of an analyzer program is optional. "URL path components for 3270 display applications" on page 195 explains how to specify the URL path correctly when a URIMAP definition is used.

CICS Web support does not provide support for partitions, logical devices codes, magnetic slot readers, outboard formatting, or other hardware features. You can use detectable fields with light pen support.

CICS Web support processing for 3270 application programs

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Figure 8 shows how CICS Web support processes a terminal-oriented transaction.

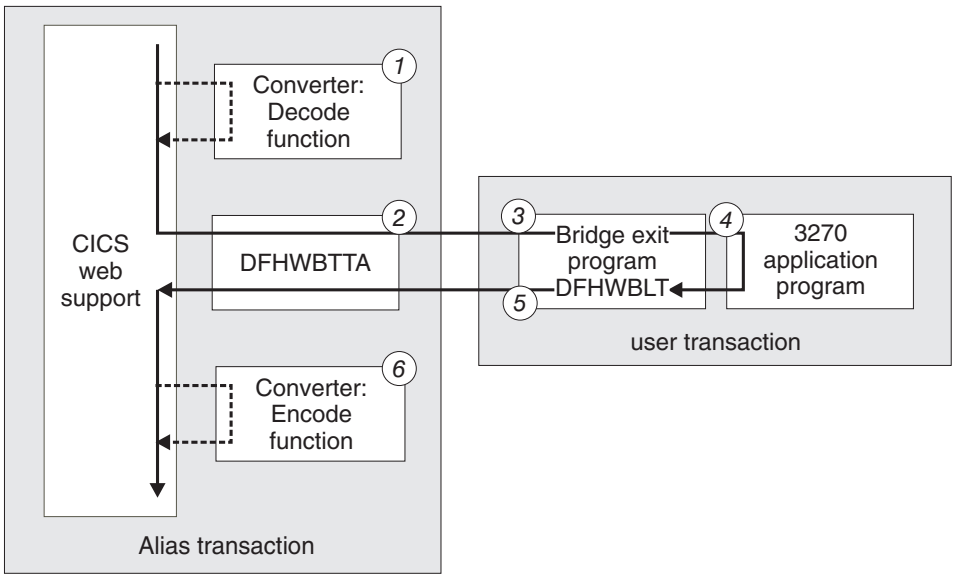


Figure 8. How CICS Web support interacts with a 3270 application program

The steps shown in the figure are:

1. Optionally, a converter program constructs the input that is passed to program DFHWBTTA.

2. DFHWBTTA attaches the user's transaction, specifying DFHWBLT as the bridge exit program, and waits for a response from DFHWBLT. The user's transaction executes in a 3270 bridge environment.
3. The bridge exit sets up a 3270 environment for the user's application program.
4. The application program processes the input, and constructs the 3270 output.
5. The bridge exit interprets the 3270 output, and passes the HTTP response to DFHWBTTA.
6. Optionally, a converter program modifies the output that is passed to the Web client.

Note: When you use CICS Web support with 3270 applications, the application program executes under its own transaction, and not under the alias transaction.

For more information about the 3270 bridge, see Bridging to 3270 transactions in the *CICS External Interfaces Guide*.

URL path components for 3270 display applications

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

To invoke a CICS 3270 application from a Web browser, you must enter a URL with a path component that starts by invoking the application program name DFHWBTTA, DFHWBTTB, or DFHWBTTC, together with an appropriate alias transaction and converter program (if required). Note that this alias transaction does not apply to the 3270 application itself, which runs under its own transaction.

Using an analyzer program
If you are using an analyzer program like the CICS-supplied sample analyzer DFHWBADX to handle requests, the path component of the URL includes the name of the application program (DFHWBTTA, DFHWBTTB or DFHWBTTC). It also includes the name of any converter program that you are using, and the name of the alias transaction for request processing (such as the default CICS-supplied alias transaction CWBA). As explained in "CICS-supplied sample analyzer program DFHWBADX" on page 135, these elements of the path are extracted by the analyzer program and used to invoke subsequent processing stages.

Using a URIMAP definition
If you are using a URIMAP definition to handle requests, the path component of the URL is specified in the PATH attribute. With URIMAP definitions, the path component of the URL does not need to include explicit information about the application program, converter program and alias transaction (although it can still do so). All these elements can be specified in the URIMAP definition, using the PROGRAM, CONVERTER, and TRANSACTION attributes. This part of the path component can then be replaced by any path of your choice. To satisfy the requirements of DFHWBTTA itself, use an asterisk as a wildcard character at the end of the path that you specify in the URIMAP definition. This allows the remainder of the path component to be varied to control DFHWBTTA.

Using both a URIMAP definition and an analyzer program
You can use an analyzer program in the processing path for a request by specifying the ANALYZER(YES) option in the URIMAP definition. The analyzer program can dynamically modify the converter program, alias

transaction ID and program name that are specified by the URIMAP definition, and DFHWBTTA can see these changes.

After providing the information needed to invoke the application program, the next part of the path component of the URL is used to provide control information to DFHWBTTA. This information includes:

- A keyword to specify if unformatted mode should be used.
- The transaction ID of the 3270 application you want to use.
- An input parameter for the specified transaction, using plus signs (+) as a delimiter.

Figure 9 shows the syntax of the path component that is interpreted by DFHWBTTA.

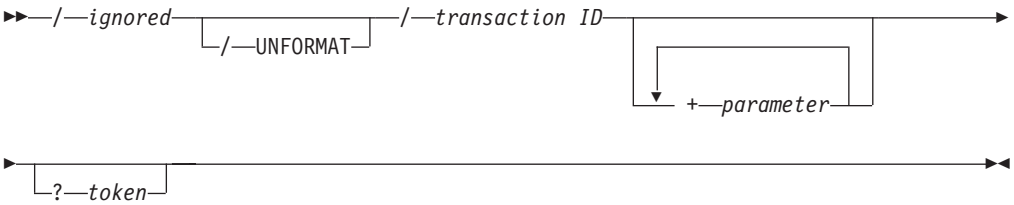


Figure 9. Syntax of the path interpreted by DFHWBTTA

DFHWBTTA interprets the path component of the URL as follows:

ignored
The first part of the path is ignored by DFHWBTTA. This is the part that is interpreted by the analyzer, or matched to a URIMAP definition, to provide the information needed to invoke the application program.

UNFORMAT
The 3270 display can operate in two modes, formatted mode, and unformatted mode. If this keyword is present, DFHWBTTA simulates a 3270 display operating in unformatted mode. If this keyword is omitted, DFHWBTTA simulates a 3270 display operating in formatted mode.

For more information about how the 3270 display operates in unformatted mode, see Unformatted mode in the *CICS Application Programming Guide*.

transaction ID
On the initial request, this information specifies the CICS transaction to be run. This element of the path is ignored on a continuation request.

parameter
Specifies an input parameter for the transaction. Use plus signs (+), not spaces, as a delimiter to separate the transaction id and this data, and between elements of this data.

token
This is ignored by DFHWBTTA. It can be used by an analyzer program.

The URL must always be coded in this form.

For example, if you are using the CICS-supplied analyzer program DFHWBADX, you could use the following URL path to issue the CEMT INQ TAS command:

```
/cics/cwba/dfhwbtta/CEMT+INQ+TAS
```

In this example:

- `cics` is used to indicate that no converter program is required.
- `cwba` is the name of the alias transaction for request processing.
- `dfhwbtta` is the name of the application program.
- `CEMT+INQ+TAS` tells DFHWBTTA to access the CEMT transaction and issue the INQ TAS command.

Alternatively, you could set up a URIMAP definition that includes the following attributes:

```
Path:        /terminal/*
Transaction:  CWBA
Program:      DFHWBTTA
```

With this URIMAP definition enabled, you could use the following URL path to issue the CEMT INQ TAS command:

```
/terminal/CEMT+INQ+TAS
```

In this example:

- `terminal` matches to the URIMAP definition, which specifies the name of the alias transaction and application program.
- `CEMT+INQ+TAS` is ignored by the URIMAP definition, but tells DFHWBTTA to access the CEMT transaction and issue the INQ TAS command.

Initial and continuation requests

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

DFHWBTTA distinguishes two types of HTTP requests by their context within a transaction: initial requests and continuation requests.

Initial requests

The initial request initiates a CICS transaction. Send the initial request in one of these ways:

- Type the URL explicitly.
- Select a link in an HTML page.
- Select a button in an HTML form. Any data entered in the form will be ignored.

Continuation requests

Continuation requests continue an existing CICS transaction. Send a continuation request in this way:

- Select a button in an HTML form that was displayed as a response to the previous request.

Continuation requests use the HTML POST method; form data is transmitted in the entity body of the HTML request.

In a conversational or pseudoconversational transaction, with several interactions between a Web client and CICS, there is one initial request, followed by one or more continuation requests. In simpler transactions, with just one interaction, there is one initial request, and no continuation request.

A hidden element (DFH_STATE_TOKEN) in the HTML form displayed by the initial request and returned by subsequent requests is used to distinguish between initial requests and continuation requests, and to associate continuation requests with the correct transaction.

The transaction ID on continuation requests

On a continuation request, the URL is coded in the form displayed by the previous request. However, the transaction ID coded in the URL is ignored on a continuation request.

Instead, the transaction is determined in the following way:

- When the continuation request is part of a conversational transaction, the same transaction continues execution.
- When the continuation request is part of a pseudoconversational transaction, then:
  - If the previous transaction ended with an **EXEC CICS** RETURN command with the TRANSID option, the specified transaction ID is the one that will be used.
  - If the previous transaction did not specify a transaction ID on its **EXEC CICS** RETURN command, but the AID is associated with a transaction ID, that transaction ID is used.
  - If no transaction ID was specified on the **EXEC CICS** RETURN command, and there is no transaction ID associated with the AID, then CICS gets the transaction ID from the HTML form.

The transaction ID in an HTML form

When a transaction is attached from a 3270 display, CICS expects to find the transaction ID in the first modified field in the 3270 data stream.

The order in which Web clients transmit form data is not always predictable, so CICS uses a mapping between the name of the form field and the corresponding position on the 3270 screen:

- For transactions that do not use BMS maps, the mapping uses the field name directly, as the name reflects the position of the field on the 3270 screen.
- For transactions that use BMS maps, the field names do not always reflect the positions on the 3270 screen, and an indirect mapping is used. The mapping makes use of the hidden variables DFH_NEXTTRANSID.*n*. When an HTML template is created from a BMS map, up to five variables are created. The value of each variable is the name of an input field, in sequence of 3270 buffer position.

  When CICS receives an HTTP request, it examines each DFH_NEXTTRANSID field in turn, to determine the name of the input field to which it refers, and whether the HTTP request contains a value for the field. If it does, then it is because the end user has modified it, and it is therefore assumed to contain the transaction ID of the next transaction.

  When a screen is constructed by merging the output from several BMS and non-BMS SEND commands, there are situations in which input fields are suppressed (see "How the footing section is chosen" on page 216 for more information). So that CICS can correctly identify the transaction ID in the 3270 data stream, you must ensure that input fields which may contain the transaction ID are not suppressed in the merged HTML page.

Terminal control commands in CICS Web support for 3270 applications

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS Web support for 3270 applications supports the following terminal control commands: SEND, CONVERSE and RECEIVE

About this task

It also supports minimum function BMS and the SEND TEXT command.

The DEFRESP option on the SEND and CONVERSE commands is ignored. This may affect application recovery.

HTML templates generated from BMS maps

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The features of the 3270 display system have many parallels with HTML forms:

- In both cases the display area can contain fixed text, and areas where the user can enter data.
- The AID keys on the 3270 keyboard have a similar function to the buttons displayed on an HTML form.
- In both cases it is possible to detect whether the end user has modified the contents of a data entry field.

Templates generated from BMS maps contain a number of elements to represent the features of the 3270 display:

- Protected fields in the map are displayed as normal HTML text.
- Unprotected fields in the map are displayed as text input elements. CICS gives each element a two-part name, which can be up to 32 characters in length:
  - The first part of the name is 11 characters long, and has the following form:

    F*rrcccllll*_

    where
    - *rr* is a two-digit number which denotes the row in which the field is displayed on a 3270 screen.
    - *ccc* is a three-digit number which denotes the column in which the field is displayed on a 3270 screen.
    - *llll* is a four-digit number which denotes the length of the field.
  - For BMS fields that are named in the map, the second part consists of the name used in the map, truncated if necessary to a length of 21 characters.
  - For BMS fields that are unnamed, the second part is of the form DFH_*nnnn* where *nnnn* is a 4–digit number. The fields are numbered sequentially as they are encountered in the BMS map.

  For example, suppose that the third unnamed and unprotected field is located at row 2 and column 11 of the screen, and has a length of 16 characters. The generated two part name is

  F020110016_DFH0003

Now suppose the same field had a name of TOTAL_MONTHLY_PURCHASES in the BMS map. The name which CICS generates for the HTML element is:

```
F020110016_TOTAL_MONTHLY_PURCHAS
```

Note: The sequence in which fields are displayed on the 3270 screen may not be the same as the sequence in which they are coded in a BMS map definition. When the corresponding template is displayed on a Web client, the fields *are* displayed in the sequence in which they are coded.

- Each attention key supported by the 3270 display is simulated as a submit button. The buttons are named:
  - DFH_PF01 through DFH_PF24
  - DFH_PA1 through DFH_PA3
  - DFH_ENTER, DFH_CLEAR

  When the end user selects one of these buttons, the corresponding variable is transmitted in the HTTP request. CICS uses the variable to determine which AID to simulate in the 3270 application.

  An additional submit button named DFH_PEN is used with detectable fields.
- Detectable fields are simulated as text elements with a preceding check box. See "Using detectable fields" on page 207 for more information.
- A hidden element (DFH_STATE_TOKEN) is used to maintain the display state seen by the application over a number of interactions with the Web client.
- A hidden element (DFH_CURSOR) and a JavaScript function (dfhinqcursor()) cooperate to return the cursor position to the application.
- A series of hidden elements (DFH_NEXTTRANSID.1 through DFH_NEXTTRANSID.*n*) are used to capture a transaction id entered in a Web client field.

HTML pages generated from 3270 data streams

For applications that do not use BMS, CICS Web support generates an HTML page which is in three parts: a heading section, a screen image section, and a footing section.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The heading section

CICS Web support generates the following heading section:

```
<!doctype html public "-//W3C//DTD HTML 3.2//EN">
<html>
<STYLE TYPE="text/css">
<!--
    TABLE, TR, TD
    { padding: 0mm    }
    TABLE
    { width: 60%    }
-->
.BRIGHT
{font-weight: bold}
{font-family: courier}
.INPUT
{font-family: courier}
</STYLE>
<head>
<title>CICS Web support screen emulation - tranid</title>
<meta name="generator" content="CICS Transaction Server/2.2.0">
```

```
<script language="JavaScript">
<!--
 function dfhsetcursor(n)
  {for (var i=0;i<document.form3270.elements.length;i++)
     {if (document.form3270.elements[i].name == n)
         {document.form3270.elements[i].focus();
          document.form3270.DFH_CURSOR.value=n;
          break}}}
 function dfhinqcursor(n)
   {document.form3270.DFH_CURSOR.value=n}
 // -->
</script>
</head>
<body onLoad="dfhsetcursor('&DFH_CURSORPOSN;')">
```

You can modify the appearance of the page by providing your own heading section. For more information, see "Modifying the output from DFHWBTTA" on page 203.

The screen image section

This section of the HTML page is generated directly from an internal representation of a 3270 screen image, whose size is determined from the DEFSCREEN and ALTSCREEN definitions on the FACILITYLIKE terminal definition associated with your transaction. It contains the following elements:

Normal HTML text
: Simulates protected fields

Text input elements
: Simulate unprotected fields. Each element is given a name which is 11 characters long, and has the following form:

  `Frrcccllll_`

  where

  - *rr* is a two-digit number which denotes the row in which the field is displayed on a 3270 screen.
  - *ccc* is a three-digit number which denotes the column in which the field is displayed on a 3270 screen.
  - *llll* is a four-digit number which denotes the length of the field.

  For example:

  - A field at row 1 and column 1 on a 3270 display, and which has a length of 78 bytes will be named

    `F010010078_`

Text elements with a check box
: Simulate detectable fields. See "Using detectable fields" on page 207 for more information.

Hidden elements
: A hidden element named `DFH_STATE_TOKEN` is used to maintain the display state seen by the application over a number of interactions with the Web client.

  A hidden element (`DFH_CURSOR`) and a JavaScript function (`dfhinqcursor()`) cooperate to return the cursor position to the application.CICS uses the JavaScript `focus()` method to position the cursor in the input box or field

specified by DFH_CURSOR. Note that focus() cannot position the cursor over a particular character within the input box or field, but only at the first character position.

```
<!doctype html public "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>CICS Web support screen emulation - tranid</title>
<meta name="generator" content="CICS Transaction Server/2.1.0">
<script language="JavaScript">
<!--
 function dfhsetcursor(n)
  {for (var i=0;i<document.form3270.elements.length;i++)
     {if (document.form3270.elements[i].name == n)
         {document.form3270.elements[i].focus();
          document.form3270.DFH_CURSOR.value=n;
          break}}}
 function dfhinqcursor(n)
   {document.form3270.DFH_CURSOR.value=n}
 // -->
</script>
</head>
<body onLoad="dfhsetcursor('&DFH_CURSORPOSN;')">
```

The HTML generated from the 3270 screen image is similar to the HTML generated in the templates for BMS maps. The horizontal and vertical alignment of information on the page is achieved using an HTML table:

- The HTML table contains one column for each different column of the 3270 screen that contains the start of a field. For example, if the 3270 screen contains fields that start in columns 2, 11, 21 and 55, then the HTML table will contain four columns. Thus, all fields whose starting positions are vertically aligned in the 3270 screen will appear vertically aligned in the HTML page.
- The HTML table contains one row for each row of the 3270 screen that contains the start of a field. Thus, all fields whose starting positions are horizontally aligned in the 3270 screen will appear horizontally aligned in the HTML page. Rows on the 3270 screen that do not contain fields are not represented in the HTML table.
- Within the table, text is displayed in a proportional font

Consider a 3270 screen containing the following fields:

Field	Row	Starting Column
Field_1	2	2
Field_2	3	2
Field_3	3	35
Field_4	4	2
Field_5	4	35
Field _6	9	2
Field_7	9	18
Field_8	9	35

All the fields start in column 2, 18, or 35 of the 3270 screen. Therefore the resulting HTML table will have three columns. Similarly, all the fields are located on row 2, 3, 4, or 9 of the 3270 screen, so the HTML table will have four rows.

You can use the encode function of a converter program to modify the screen image section. For more information, see “Using a converter program with DFHWBTTA” on page 206.

The footing section

CICS Web support generates the following footing section. Each attention key supported by the 3270 display is simulated as a submit button. When the end user selects one of these buttons, the corresponding variable is transmitted in the HTTP request. CICS uses the variable to determine which AID to simulate in the 3270 application. An additional submit button named DFH_PEN is used with detectable fields.

```
<input type="submit" name="DFH_PF1"  value="PF1">
<input type="submit" name="DFH_PF2"  value="PF2">
<input type="submit" name="DFH_PF3"  value="PF3">
<input type="submit" name="DFH_PF4"  value="PF4">
<input type="submit" name="DFH_PF5"  value="PF5">
<input type="submit" name="DFH_PF6"  value="PF6">
<input type="submit" name="DFH_PF7"  value="PF7">
<input type="submit" name="DFH_PF8"  value="PF8">
<input type="submit" name="DFH_PF9"  value="PF9">
<input type="submit" name="DFH_PF10"  value="PF10">
<input type="submit" name="DFH_PF11"  value="PF11">
<input type="submit" name="DFH_PF12"  value="PF12">
<br>
<input type="submit" name="DFH_PF13"  value="PF13">
<input type="submit" name="DFH_PF14"  value="PF14">
<input type="submit" name="DFH_PF15"  value="PF15">
<input type="submit" name="DFH_PF16"  value="PF16">
<input type="submit" name="DFH_PF17"  value="PF17">
<input type="submit" name="DFH_PF18"  value="PF18">
<input type="submit" name="DFH_PF19"  value="PF19">
<input type="submit" name="DFH_PF20"  value="PF20">
<input type="submit" name="DFH_PF21"  value="PF21">
<input type="submit" name="DFH_PF22"  value="PF22">
<input type="submit" name="DFH_PF23"  value="PF23">
<input type="submit" name="DFH_PF24"  value="PF24">
<br>
<input type="submit" name="DFH_PA1"   value="PA1">
<input type="submit" name="DFH_PA2"   value="PA2">
<input type="submit" name="DFH_PA3"   value="PA3">
<input type="submit" name="DFH_CLEAR" value="Clear">
<input type="submit" name="DFH_ENTER" value="Enter">
<input type="submit" name="DFH_PEN"   value="Pen">
<input type="reset"  value="Reset">
</form>
</body>
</html>
```

You can modify the appearance of the page by providing your own footing section. For more information, see “Modifying the output from DFHWBTTA.”

Modifying the output from DFHWBTTA

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

You can modify the output from DFHWBTTA either by customizing the HTML or by providing your own heading and footing sections.

About this task

For applications that use BMS, you can customize the HTML templates created from BMS maps. For more information on customizing HTML templates, see "Generating customized HTML templates" on page 211.

For non-BMS applications, and BMS applications invoked using DFHWBTTC, you can modify the appearance of the page by providing your own heading and footing sections. You cannot change the screen image section directly, although tags which you insert in the heading section may affect the appearance of the following sections.

To provide your own heading and footing sections, define and install one or more of the following templates, whose names are defined in the TEMPLATENAME fields of DOCTEMPLATE definitions:

*tran***HEAD**
: This template is inserted at the head of the HTML page that is the output for transaction *tran*, if it is installed.

CICSHEAD
: This template is inserted at the head of the HTML page that is output for transactions which do not have a corresponding *tran*HEAD template installed.

*tran***FOOT**
: This template is inserted at the foot of the HTML page that is output for transaction *tran*, if it is installed. If this template is not installed, CICSFOOT is used instead.

CICSFOOT
: This template is inserted at the foot of the HTML page that is output for transactions which do not have a corresponding *tran*FOOT template installed.

For more information about creating document templates, see Programming with documents and document templates in the *CICS Application Programming Guide*.

The heading section generated by CICS Web support, including DFHWBTTC, uses the EBCDIC Latin character set (code page 037). If you use a different code page in your CICS system, you must create a similar heading section, using your own code page:

1. Create a document template called CICSHEAD containing your heading section.
2. Define and install a DOCTEMPLATE definition for the template.

. The following characters used in the CICS-generated heading section have different representations in code pages other than 037:

```
! [ ] { }
```

Supplying your own heading template

If you supply your own heading template, you must supply some of the required elements of an HTML page.

About this task

A heading template should contain the following HTML elements:

- A doctype tag. For example:

  ```
  <!doctype html public "-//W3C//DTD HTML 3.2//EN>
  ```

- An <html> tag
- A <head> tag
- A <STYLE> tag, which must contain style sheet rules for the BRIGHT and INPUT classes. For example:

  ```
  <STYLE TYPE="text/css">
  <!--
      TABLE, TR, TD
      { padding: 0mm    }
      TABLE
      { width: 60%    }
  -->
  .BRIGHT
  {font-weight: bold}
  {font-family: courier}
  .INPUT
  {font-family: courier}
  </STYLE>
  ```

  You can use the width attribute of the TABLE element to fine-tune the appearance of the screen image section.

- A </head> tag
- A <body> tag. You can use this tag to specify text colors, or an image to be used as the background for the page. For example:

  ```
  <body background="/dfhwbimg/background2.gif" bgcolor="#FFFFFF"
   text="#000000" link="#00FFFF" vlink="#800080" alink="#FF0000"
  onLoad="dfhsetcursor('&DFH_CURSORPOSN;')">
  ```

  Note: This example uses DFHWBIMG which is described in "Using DFHWBIMG to display graphics" on page 208.

- Optionally any other HTML elements you need to customize the page.

Supplying your own footing template

If you supply your own footing template, you must supply some of the required elements of an HTML page.

About this task

A footing template should contain the following HTML elements:

- Input buttons to represent any programmed function keys or the ENTER key. For example:

  ```
  <input type="submit" name="DFH_PF1"  value="Help">
  <input type="submit" name="DFH_PF3"  value="Quit">
  <input type="submit" name="DFH_ENTER"  value="Continue">
  ```

  These form part of the HTML form begun by CICS. The buttons, when selected by the user, produce the AID indicator discussed in "HTML pages generated from 3270 data streams" on page 200, so should have the names described there. The *value* parameter specifies the legend that appears on the generated button. It is not used by DFHWBTTA.

- A </form> tag

- Optionally any other HTML elements you need to customize the page.
- A </body> tag to close the page
- An </html> tag

Using a converter program with DFHWBTTA

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

You can use the decode function of the converter program to modify requests passed to DFHWBTTA.

About this task

- When an HTML form is submitted by the client using one of the buttons that represent an attention key, the request contains a field indicating which button was selected. You can simulate the effect of a different attention key by modifying the request. Change the value of the field to the desired attention key, or insert a new field after the one transmitted by the Web client.
- When an HTML form is submitted by the client, the DFH_CURSOR field contains the name of the field that contains the cursor. You can simulate the effect of a different cursor position by modifying the request. Change the value of the DFH_CURSOR field to contain a different field name, or insert a new DFH_CURSOR field after the one transmitted by the Web client.
- You can select the next transaction ID by changing the DFH_NEXTTRANSID.*n* variables in the continuation request. You can insert or delete a variable, or change the value of one of them. For more information about how these fields are used to determine the next transaction ID, see "The transaction ID in an HTML form" on page 198.

Do not modify the value of DFH_STATE_TOKEN.

You can use the encode function of the converter program to modify the output from DFHWBTTA:

- The response is in a buffer that begins with a 32-bit unsigned number that specifies the length of the buffer. The rest of the buffer is the HTTP response. The HTML in the response is that corresponding to the output BMS map or 3270 data stream from the transaction program.
- The HTTP headers in the HTTP response are generated automatically by DFHWBTTA. The headers generated by DFHWBTTA are:
  - Content-type: text/html
  - Content-length: <length of the entity body>
  - Pragma: no-cache
  - Connection: Keep-Alive (if this is an HTTP 1.0 persistent connection)

  If any additional headers are required, the Encode function of the converter should be used to add them to the HTTP response.

Enabling detectable fields

To enable detectable field processing over the CICS Web support 3270 bridge, you must define a bridge facility with light pen support enabled.

About this task

To do this, follow these steps:

Procedure

1. Copy the following definitions to a new group. Unless all applications running on the CICS system require light pen support, you should also rename both definitions:
   - The CICS-supplied bridge facility CBRF, in group DFHTERM.
   - Its default TYPETERM, DFHLU2, in group DFHTYPE.
2. In the TYPETERM definition, change the LIGHTPEN option under "DEVICE PROPERTIES" to YES.
3. In the TERMINAL definition, change the TYPETERM parameter to point to the new TYPETERM.
4. Install the definitions in the CICS region.
5. If you have created a new bridge facility definition, update the PROFILE definition of the 3270 transaction which you are going to run with CICS Web support, so that the bridge facility will be modelled on the new TERMINAL/TYPETERM definition:
   a. Identify the PROFILE that the transaction uses by using CEDA to view the PROFILE parameter of the TRANSACTION definition.
   b. If the profile is a CICS-supplied profile, make a copy of it to your own group and rename it.
   c. Alter the new PROFILE and enter the name of your new bridge facility in the FACILITYLIKE parameter.
   d. Alter your TRANSACTION definition to use the new PROFILE definition.

Using detectable fields

When CICS generates an HTML page from the 3270 data stream, it simulates detectable fields with a text input field preceded by a check box.

Before you begin

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

To use detectable fields, the bridge facility associated with the transaction must be configured. “Enabling detectable fields” on page 206 tells you how to do this.

About this task

Detectable fields are those in which:

- The field attribute byte identifies the field as being detectable or intensified.
- *and* The first character of the 3270 field contains a valid designator character. This can be an ampersand (&), a right angle bracket (>), a question mark (?), a blank, or a null.

For more information about detectable fields, see Field selection features in the *CICS Application Programming Guide*.

When the check box and text input field are displayed on the Web client:

- The designator character in the 3270 field is not displayed. Accordingly, the field length in the Web client is one character shorter than it is in the 3270 datastream.
- If the designator character is a right angle bracket (>), the check box contains a check symbol (✓). Otherwise, the check box is empty.

To use the detectable field on the Web client:

Procedure

- Check the check box to simulate setting the modified data tag (MDT) bit in the 3270 data stream. Uncheck the box to set the modified data tag off. Entering data in the text field in the HTML page *does not* change the modified data tag.
- To transmit data to the CICS application, check the check box , and select the button named `DFH_PEN`.
  - If just one attention field is checked, the CICS application receives the contents of just that field. The EIBAID field is set to DFHPEN.
  - If several attention fields are checked, the CICS application receives the contents of the field closest to row 1 and column 1 of the 3270 screen. The EIBAID field is set to DFHPEN.
  - If no attention fields are checked, CICS receives the contents of all the fields. The EIBAID field is set to DFHENTER.

Using DFHWBIMG to display graphics

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

CICS supplies the following graphics that you can use in your Web applications.

About this task

CICS.GIF
The CICS logo

MASTHEAD.GIF
The CICS logo with the text 'CICS Web Interface'

BACKGROUND1.GIF
A background containing the characters 'CICS'

BACKGROUND2.GIF
A background containing the characters 'CWI'

TEXTURE1.JPEG
A textured background

TEXTURE2.JPEG
A textured background

TEXTURE3.JPEG
A textured background

TEXTURE4.JPEG
A textured background

TEXTURE5.JPEG
A textured background

TEXTURE6.JPEG
A textured background

To display the graphics on your Web browser, enter a URL in which (after translation to upper case) the path is in this form:

```
/DFHWBIMG/filename
```

where **filename** is the name of one of the graphics listed. For example:

```
 /DFHWBIMG/Texture1.jpeg
```

To incorporate any of the graphics in your output, include the path in the appropriate HTML tag. For example, you can include a textured background with the following tag:

```
<body background="/DFHWBIMG/background1.gif" ... >
```

CICS processes HTTP requests in which the path begins with "/DFHWBIMG" as a special case; the analyzer is not called, and DFHWBIMG runs as the converter program.

CICS uses some of these graphics in the templates used for CICS-supplied transactions.

The graphics which CICS supplies are hard coded as part of DFHWBIMG and are not available as separate files; DFHWBIMG does not support the display of graphics apart from those named.

Chapter 16. Creating HTML templates from BMS definitions

If you want to create an HTML template from an existing BMS map set for which you do not have the source code, you may be able to reconstruct the source from the corresponding load module.

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Use the BMS macro generation utility program (DFHBMSUP), which is described in BMS macro generation utility in the *CICS Operations and Utilities Guide*.

CICS provides catalogued procedure DFHMAPT for installing HTML templates which have been created from a BMS map set. See Installing map sets and partition sets in the *CICS Application Programming Guide* for details.

About BMS-generated templates

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

A template generated from a BMS map contains: constants and input fields, buttons, hidden variables, a JavaScript function and a JavaScript exception handler.

- Constants and input fields from the map
- Buttons to represent the following:
  - ENTER and CLEAR keys
  - PA1, PA2, and PA3 keys
  - Program function keys PF1 to PF24
  - HTML reset
- Up to five hidden variables, `DFH_NEXTTRANSID.1` to `DFH_NEXTTRANSID.5`, whose values are the names of the first five fields in the map. The use of these variables is explained in Chapter 15, "CICS Web support and 3270 display applications," on page 193.
- A hidden variable `DFH_CURSOR` whose value is the name of the field in which the cursor is set in the map. If the cursor is located in an unnamed field, `DFH_CURSOR` is zero.
- A JavaScript function `dfhsetcursor()`. When `DFH_CURSOR` contains the name of a field, the function sets the cursor position to that field.
- A JavaScript exception handler for the `onFocus` exception. This function invokes `dfhsetcursor`, and tracks the movement of the cursor.

Generating customized HTML templates

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

There are three ways that you can customize HTML templates generated from BMS maps.

About this task

- You can modify the way HTML templates are generated from BMS maps by coding your own version of the DFHMSX macro.
- You can add HTML text to the generated map by using the DFHWBOUT macro within the BMS map definitions
- You can manually edit the generated HTML. This is useful when:
  - you want to override the dynamic changes to attributes which take place when a program issues a MAP SEND command
  - you want to use the HTML template outside the Web 3270 environment.

  In both cases, you will need to change the *Frrcccllll* variables which are added by the template generation process.

You are strongly recommended to avoid editing CICS-generated HTML templates unless all your SEND MAP commands use the ERASE option. SEND MAP commands without ERASE will result in merging of HTML during CICS runtime. Runtime logic is expecting to encounter HTML which was generated by the CICS template generator. In particular you should avoid making changes to `<tr>` tags.

For examples of customized templates, see "Customization examples" on page 224.

CICS provides HTML templates for the following CICS-supplied transaction that uses BMS:

CETR

The templates provided use the EBCDIC Latin character set (code page 037). If you use a different code page in your CICS system, you must generate your own version of these templates: the following characters used in the CICS-generated heading section have different representations in code pages other than 037:

```
! [ ] { }
```

Use the **CODEPAGE** parameter on the DFHMDX macro to specify the code page.

Customizing with the DFHMSX macro

You can modify the way HTML templates are generated from BMS maps by coding your own version of the DFHMSX macro.

About this task

You can specify:

- the 3270 keys that are represented by buttons
- the text or image that is displayed on each button
- the title of the HTML page
- a masthead graphic to be displayed at the top of the HTML page
- the page background as a graphic file or color
- the color of normal text, unvisited links, visited links and active links
- whether the page should include an HTML reset button, and the text displayed on it
- a mapping between the colors used in the BMS map and the colors used for the corresponding text in the HTML template
- which BMS fields should be suppressed from the HTML page
- JavaScript `onLoad()` and `onUnload()` exception handlers

- whether the text in the template is to be displayed in a proportional or non-proportional font
- the code page to be used when the template is generated, and the code point to be used for the special characters [] { } and !
- that protected fields should be right-aligned in the HTML page

Note:

1. The ATTRB=BRT option of a BMS field has no effect for an unnamed, unprotected (input) field.
2. DFHBMEOF, a 3270 attribute bit of the attribute byte of a field named in the logical map, is not set if the field is emptied (for example, with the DEL key), or if the field was already empty (nulls or spaces) on the previous SEND command and that field's Modified Data Tag (MDT) was off.

When you code your own version of the DFHMSX macro, you can specify that the options you code apply to:

- All maps in all map sets
- All maps in certain map sets
- Individual maps

Installing the HTML templates

About this task

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The procedure is as follows:

Procedure

1. Review your CICS application programs and their use of BMS to see if customization is necessary.
2. For the applications that need customized HTML pages, create a customization macro definition, and store it in a library in the concatenation of macro libraries specified in the SYSLIB DD statement for the assembler. Write appropriate DFHWBOUT macro invocations, and put them in the appropriate places in your map definitions.
3. Assemble the existing map definitions with TYPE=TEMPLATE on the DFHMSD macro, or SYSPARM=TEMPLATE in the parameters passed to the assembler. Note that the label on the DFHMSD macro is used to name the HTML templates produced for each map in the map set being processed. The HTML template names consist of the label from the DFHMSD macro, followed by a one- or two-character suffix generated with the characters A-Z and 0–9. The two-character suffix is used when there are more than 36 maps in the mapset, and in this case the mapset name must be six characters or less. For the bridge exit to match the HTML template with the BMS map when a BMS SEND or RECEIVE is issued by a program, the HTML template members must match the name of the map set value used on the SEND and RECEIVE statements. If you are using a customizing macro, you must add the name of the customizing macro to the TYPE. The assembler produces IEBUPDTE source statements that set up one template for each map in a map set.
4. Use IEBUPDTE to store the templates in the template library. If the record format of the template library is not fixed blocked, you will need to store them

in another partitioned data set, and then convert them to the record format of the template library using, for instance, ISPF COPY.

5. If you want to put your templates in a partitioned data set other than the one specified in the DFHHTML DDname, you must define DOCTEMPLATE definitions for your templates, and specify an alternate DDname. The alternate DDname must also be specified in your CICS JCL.

   To allocate a partitioned data set containing templates to a specific DD name in order to install templates from it, you can use the ADYN sample transaction. First install the DFH$UTIL group, which contains ADYN and its related programs, then run ADYN:

   ```
   ADYN
   ALLOC DDNAME(ddname) DATASET('template-pds') STATUS(SHR)
   ```

   where *ddname* is the DDname specified in the DOCTEMPLATE definition, and *template-pds* is the name of the partitioned data set containing the template to be installed. For further information on installing and using ADYN, see the *CICS Customization Guide*.

Results

Size restrictions of HTML templates

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

There is no restriction on the size of templates used by transactions run using the 3270 Bridge. However, templates that exceed 32K of storage are processed differently from smaller templates.

About this task

To process templates larger than 32K, you must specify a path that maps to a program name of DFHWBTTB in the HTTP request. See "URL path components for 3270 display applications" on page 195 for more information.

Note that templates that require less than 32K of storage can expand to greater than 32K if symbol substitution significantly increases the amount of data.

When the template is generated, DFHWBTLG issues a message containing the amount of storage required for each template to be read from the DFHHTML data set. Use these messages to determine if you should use a program name of DFHWBTTA or DFHWBTTB.

Writing a customizing macro definition

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

You have to supply a complete assembler macro definition that is invoked by CICS-supplied assembler macros. The definition of a customizing macro must be written according to the rules for assembler macro definitions. The macro invocations in the definition must also follow the rules for assembler language macro statements.

About this task

A customizing macro definition contains the following elements:

1. A MACRO statement to begin the definition.
2. The name of the macro.
3. Any number of invocations of the DFHMDX macro.

   The syntax of DFHMDX is described in "The DFHMDX macro" on page 218, and its use is described in "Customization examples" on page 224. DFHMDX is invoked from within DFHMSX.
4. A MEND statement to end the definition.

Handling white space

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

When customizing a macro definition, the HTML specifications for white space must be taken into consideration. For 3270 terminals, blanks (EBCDIC X'40') and nulls (EBCDIC X'00') can be used to format screen data positions. When such a data stream is converted into HTML, the client interpretation of this generates different output to that found on a 3270 terminal.

About this task

A string of blanks is ignored by a client if it immediately follows a start tag, and any subsequent sequence of contiguous blanks is interpreted as one blank. To force the rendering of all blanks, you can use the `<pre>` and `</pre>` tags.

The handling of null characters is unspecified, and clients handle them inconsistently. They may or may not be displayed.

Combining BMS and non-BMS output

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

A transaction can issue a series of BMS and non-BMS commands in order to build the contents of the 3270 display screen.

This topic explains how the output from all the commands is combined to construct the HTML page which is displayed on the Web browser. The steps are:

1. When a BMS or non-BMS SEND command is issued, an HTML page (containing a heading section, a screen image section, and a footing section) is generated, but not sent to the Web client.
2. When the transaction issues a RECEIVE command, or terminates:
   - A heading section is selected from one of those which was generated previously.
   - A new screen image section is created by merging all those which were generated previously.
   - A footing section is selected from one of those which was generated previously.
3. The resulting HTML page is sent to the Web client.

How the heading section is chosen

The heading section is chosen from among the HTML pages, based upon their starting positions, and the sequence in which they were created.

1. The pages which have a starting position closest to the first row of the screen are selected.
2. If more than one page remains in the selection process, the starting position of the remaining pages is compared again. This time, the pages which have a starting position closest to the first column of the screen are selected.
3. Finally, if more than one page remains, the earliest page generated is selected.

The heading section from the remaining selected page is used in the HTML page that is sent to the Web client.

Note: In this description:

- The starting position of an HTML page generated from a BMS map is the row and column of the upper left corner of the map.
- The starting position of an HTML page generated from a non-BMS command is the upper left corner of the screen (row 1, column 1).

How the footing section is chosen

The footing section is chosen from among the HTML pages, based upon their ending positions, and the sequence in which they were created.

1. The pages which have an ending position closest to the last row of the screen are selected.
2. If more than one page remains in the selection process, the ending position of the remaining pages is compared again. This time, the pages which have an ending position closest to the last column of the screen are selected.
3. Finally, if more than one page remains, the latest page generated is selected.

The footing section from the remaining selected page is used in the HTML page that is sent to the Web client.

Note: In this description:

- The ending position of an HTML page generated from a BMS map is the row and column of the lower right corner of the map.
- The ending position of an HTML page generated from a non-BMS command is the lower right corner of the screen.

How the screen image sections are merged

When the screen image sections created as a result of a series of BMS and non-BMS SEND commands are merged, a new screen image section is created; it contains, as far as possible, all the fields from all the screen image sections which were used to construct it.

However, if fields from two or more of the constituent screen images wholly or partly overlap, this is not possible, and some of the overlapping fields may be modified or suppressed entirely:

- When fields overlap, fields associated with the earlier BMS or non-BMS SEND commands will be modified or suppressed in favor of fields from later commands.
- If an input field is partially or wholly overlapped, the entire input field is discarded and will not appear in the final HTML.

- If an input field partially overlaps some normal text, any visible text up to the start of the input field will be visible in the final HTML and the remaining data will be discarded, irrespective of whether there is more text visible after the end of the input field on a 3270 device.
- If the table cell contains a horizontal rule tag (<hr>), overlapping the contents of the cell will produce unpredictable results.

These rules are summarized in Table 7.

Table 7. Overlapping fields in a merged screen image section

Field from earlier SEND	Field from later SEND	Result
Input (unprotected)	Input (unprotected) or Text (protected)	The earlier field is entirely suppressed
Text (protected)	Input (unprotected)	Based upon the character position in the 3270 screen: • Protected characters to the left of the input field are retained. • Protected characters overlain by the input field are suppressed. • Protected characters to the right of the input field are suppressed.
Text (protected)	Text (protected)	Based upon the character position in the 3270 screen, characters from the later send will overwrite characters from the earlier send.

You can edit HTML templates created from BMS maps before using them in an application program. However, the algorithm by which the screen image sections are merged requires that the HTML in the sections has a particular structure. Therefore, when you edit the screen image section in a template, and the section will be merged with others, there are guidelines which you must follow:

- Each HTML page contains two comments with the strings DFHROW and DFHCOL respectively. The values that follow these strings are important during the merging process as they are used to calculate the position of each field on the 3270 screen. If these comments are modified or deleted, the screen image sections will not be merged but will appear in the final HTML page appended one below the other.
- The closing tags for table cells (</td>) and table rows (</tr>) are optional.
- Table cells must either contain a piece of normal text with or without additional attribute tags or they must contain an input field. Additionally, they may contain a mixture of text and input fields in the same table cell if the text and input fields follow each other without additional tags between them.
- Empty table cells may not contain null values (X'00') or spaces between the opening and closing tags. In other words, empty cells must be coded as <td></td>.
- You can bound a section of text or an input field with one or more of the following pairs of tags:

emphasis
 ...

strong ...

font ...

underline
<u> ... </u>

blink <blink> ... </blink>

If you use these tags, you must ensure that each tag has a corresponding closing tag. You must also ensure that the opening and closing tags are properly nested. For example, <u> ... </u> is properly nested, but <u> ... </u> is not.

- If you insert other tags into the table cell, they must appear before or after the text or input field but may not appear both before and after in the same table cell.
- HTML comments are allowed in the table cell and may appear before, after or on both sides of a piece of normal text or an input field.
- If the cell contains comments, it must also contain either a piece of normal text or an input field.

The DFHMDX macro

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The DFHMDX macro is invoked from within DFHMSX.

Its syntax is shown in Figure 10 on page 219.

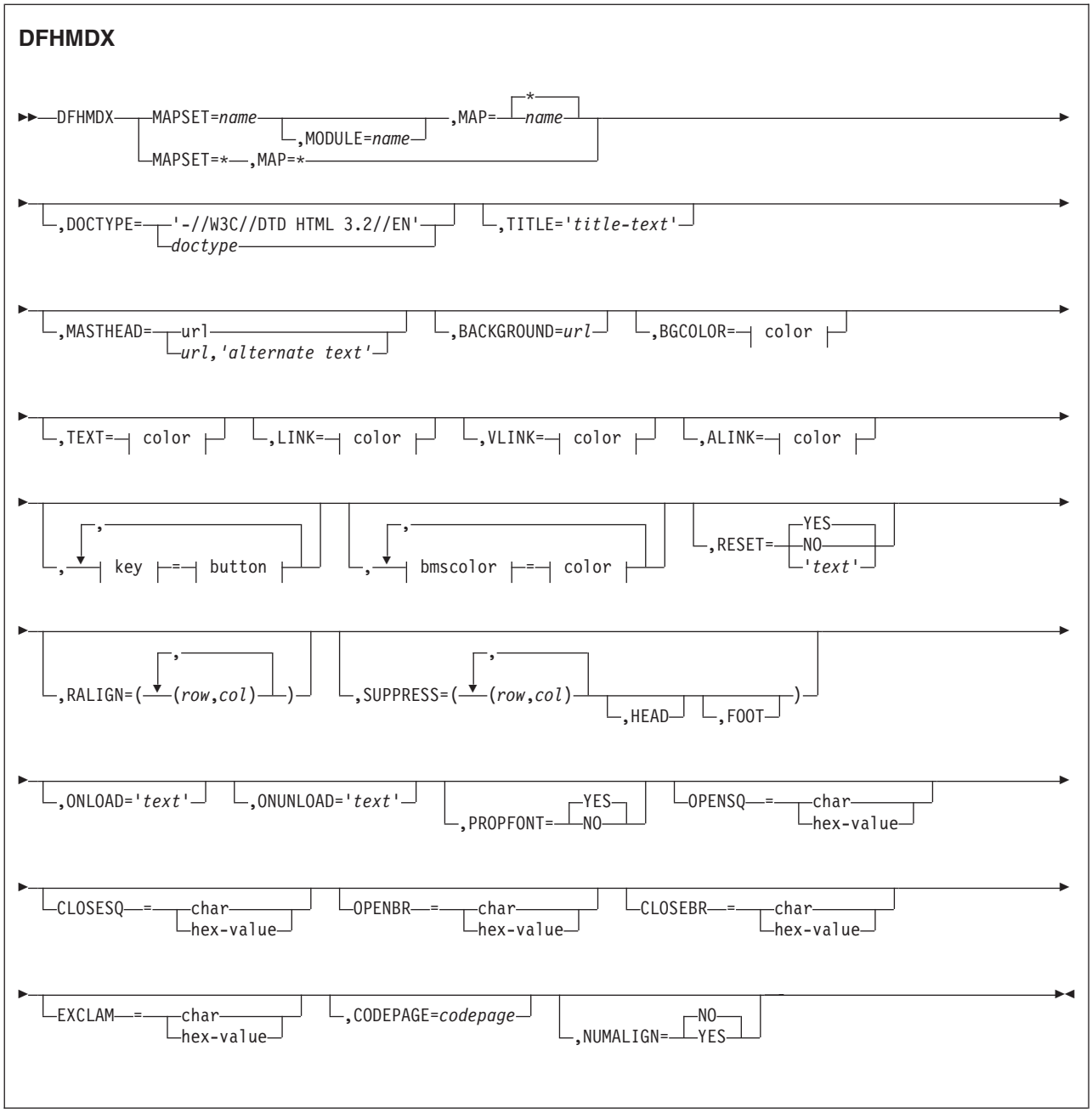


Figure 10. Syntax of DFHMDX

The keyword parameters to this macro can appear in any order.

MAPSET
specifies the name of the map set that contains the map to which other options refer. If you specify an asterisk, the options become the default to all subsequent map sets.

MODULE
specifies the name of the load module into which the map set is link-edited. You can only use this parameter if you do not specify MAPSET=*. The name you specify (which can only be seven characters) is used to construct the names of the templates by adding a single character suffix. The default value is the name of the map set.

MAP specifies the name of the map within the map set specified in MAPSET to which the options refer. If you specify an asterisk, the options become the default to all subsequent maps.

DOCTYPE
specifies the DTD public identifier part of the <!doctype> tag that you want to appear in the HTML template. The default is `-//W3C//DTD HTML 3.2//EN`, which specifies HTML 3.2. Level 3.2 is required for the color support in certain HTML tags.

TITLE specifies the title to be used as the HTML title, and as the content of the first `<h1>` tag.

MASTHEAD
specifies the URL of a masthead graphic to appear at the head of a page before the first <h1> tag. If you supply *alternate-text*, the client will use the text if it cannot load the specified graphic.

BACKGROUND
specifies the URL of a graphic file for the page background.

BGCOLOR
specifies the color of the page background.

TEXT specifies the color of normal text.

LINK specifies the color of unvisited hypertext links on the page.

VLINK
specifies the color of visited hypertext links on the page.

ALINK
specifies the color of activated hypertext links on the page.

PF1-PF24
specifies the name or image to be assigned to the simulated button for the corresponding 3270 program function key.

PA1-PA3
specifies the name or image to be assigned to the simulated button for the corresponding 3270 program attention key.

CLEAR
specifies the name or image to be assigned to the simulated button for the 3270 Clear key.

ENTER
specifies the name or image to be assigned to the simulated button for the 3270 Enter key.

PEN specifies the name or image to be assigned to the simulated button for pen selection.

BLUE specifies the color to appear in the HTML page where blue is specified in the BMS map. The default is #0000FF.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

GREEN
specifies the color to appear in the HTML page where green is specified in the BMS map. The default is #008000.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

NEUTRAL
specifies the color to appear in the HTML page where neutral is specified in the BMS map. The default is #000000.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

PINK specifies the color to appear in the HTML page where pink is specified in the BMS map. The default is #FF00FF.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

RED specifies the color to appear in the HTML page where red is specified in the BMS map. The default is #FF0000.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

TURQUOISE
specifies the color to appear in the HTML page where turquoise is specified in the BMS map. The default is #00FFFF.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

YELLOW
specifies the color to appear in the HTML page where yellow is specified in the BMS map. The default is #FFFF00.

Restriction: DFHMDX will only override the color of unnamed fields; it leaves named fields unchanged.

RESET
specifies whether the HTML reset function is to be supported. Specify YES to get a default reset button with the default legend Reset. Specify NO to get no reset button. Specify your own text for a reset button with your own legend.

RALIGN
specifies BMS map fields in which data is to be right aligned in the HTML page. The values *rr* and *cc* specified must correspond to the POS=(*rr*,*cc*) specification on the DFHMDF macro for a field to be right aligned. Each pair must be enclosed in parentheses, and the whole list of pairs must be enclosed in parentheses. If you want to right align every qualifying field which ends in a particular column, specify the end column number and put an asterisk for the row specification. Calculate the end column number for a field by adding its start column number to its LENGTH, as defined in the DFHMDF macro. Fields will be right aligned only if they are protected, unnamed, and are initialized with an INITIAL, XINIT or GINIT value in the DFHMDF macro. The RALIGN parameter is ignored if you specify it with MAP=* or MAPSET=*.

If you want to specify a list that exceeds the assembler's limit of 256 characters for a character string in macro definitions, code extra DFHMDX macros with the same MAPSET and MAP values, and put more values in the RALIGN parameters.

SUPPRESS
specifies BMS map fields that are not to appear in the HTML page. Specify any number of row and column pairs for the start positions of the fields to be suppressed. The values *rr* and *cc* specified must correspond to the POS=(*rr*,*cc*) specification on the DFHMDF macro for a field to be suppressed. Each pair must be enclosed in parentheses, and the whole list of pairs must be enclosed in parentheses. If you want to suppress all the fields in a row, specify the row number and put an asterisk for the column specification. The SUPPRESS parameter is ignored if you specify it with MAP=* or MAPSET=*.

Use the keyword HEAD to suppress the heading section in the template. Use the keyword FOOT to suppress the footing section in the template.

If you want to specify a list that exceeds the assembler's limit of 256 characters for a character string in macro definitions, code extra DFHMDX macros with the same MAPSET and MAP values, and put more values in the SUPPRESS parameters.

ONLOAD
specifies the JavaScript text to be used to replace the standard `onLoad` exception handler for the HTML page. The text must not contain double quotes ("), and single quotes (') must be doubled ('') following the usual assembler language conventions. If you use this parameter you will suppress the setting of the cursor to the field indicated by DFH_CURSOR provided by the standard `onLoad` exception handler. You can use the function `dfhsetcursor` to set the cursor position.

ONUNLOAD
specifies the JavaScript text to be used as the `onUnload` exception handler for the HTML page. The text must not contain double quotes ("), and single quotes (') must be doubled (''), following the usual assembler language conventions.

PROPFONT
specifies the font. If YES, the template will specify that text is to be presented in a proportional font, and consecutive spaces are to be reduced to a single space. If NO, the template will specify that text is to be specified in a font of fixed pitch, and consecutive spaces are to be preserved.

OPENSQ
The hex value or the character to be used to display an open square bracket. The default is X'BA' (code page 37).

CLOSESQ
The hex value or the character to be used to display a close square bracket. The default is X'BB' (code page 37).

OPENBR
The hex value or the character to be used to display an open brace. The default is X'C0' (code page 37).

CLOSEBR
The hex value or the character to be used to display a close brace. The default is X'D0' (code page 37).

EXCLAM
The hex value or the character to be used to display an exclamation mark. The default is X'5A' (code page 37).

CODEPAGE
specifies the IBM code page number in which any text generated by the template generation process is encoded. This code page must match the code page used when the templates are used by CICS, either in the HOSTCODEPAGE option of the **EXEC CICS** DOCUMENT command, or in the SRVERCP option of the DFHCNV macro selected by the analyzer program.

The standard CICS form of a host code page name consists of the code page number (or more generally CCSID) written using 3 to 5 decimal digits as necessary then padded with trailing spaces to 8 characters. For code page 37, which is fewer than 3 digits, the standard form is 037. CICS accepts any decimal number of up to 8 digits (padded with trailing spaces) in the range 1 to 65535 as a code page name, even if it is not in the standard form.

The CODEPAGE parameter must specify an EBCDIC-based code page if any symbol processing is required, as the delimiters used for symbol and symbol list processing are assumed to be in EBCDIC.

The default code page is 037.

NUMALIGN
specifies how fields that are explicitly defined as numeric in the DFHMDF macro are aligned within the table cells in the HTML template:

NO specifies that numeric fields are not right aligned within their table cells. This is the default.

YES specifies that numeric fields are right aligned within their table cells:

- For a protected field, the generated HTML text is right aligned within the cell. If the text contains trailing blanks, they may not be preserved: some clients will replace them with a single blank.

  Note: The RALIGN parameter preserves trailing blanks; the NUMALIGN parameter does not. If both parameters apply to a field (that is, if a numeric field is identified by the RALIGN parameter, and NUMALIGN=YES is specified), trailing blanks are not preserved.
- For an unprotected field, the HTML text input element (but *not* the text within the element) is right aligned within the cell.

color can be an explicit specification #*rrggbb*, where *rr*, *gg*, and *bb* are 2-digit hexadecimal numbers giving the intensities of red, green, and blue in the requested color, or it can be any one of the following color names: `AQUA`, `BLACK`, `BLUE`, `FUCHSIA`, `GRAY`, `GREEN`, `LIME`, `MAROON`, `NAVY`, `OLIVE`, `PURPLE`, `RED`, `SILVER`, `TEAL`, `WHITE`, `YELLOW`.

key can be any of `PF1` to `PF24`, `PA1` to `PA3`, `CLEAR`, `ENTER`, and `PEN`.

button can be `(IMAGE,`*url*`)`, where *url* specifies the URL of a graphic image to be used for the button, or `'`*text*`'`, where *text* is the text to be put in the button, or `NO` if the button is not to appear.

bmscolor can be any of `BLUE`, `GREEN`, `NEUTRAL`, `PINK`, `RED`, `TURQUOISE`, and `YELLOW`.

Customizing templates with the DFHWBOUT macro

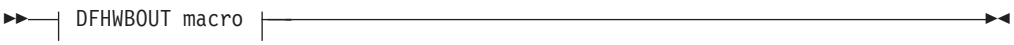
Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The DFHWBOUT macro is used to add text to the HTML page generated from a BMS map. The text appears only as part of the HTML page. If the macro is used before the first occurrence of DFHMDF in a map, the text is placed in the <head> section of the HTML page. If the macro is used elsewhere in the map, the text is placed immediately following the text generated by the preceding DFHMDF macro.

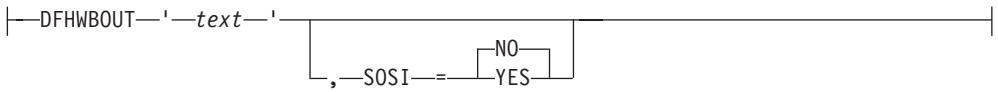
About this task

Do not use the DFHWBOUT macro when the application program builds the screen display using multiple BMS maps.

DFHWBOUT



DFHWBOUT macro



The parameters of this macro are as follows:

text The text that is to be inserted in the HTML page.

SOSI Whether the text contains DBCS characters delimited by shift-out (X'0E') and shift-in (X'0F'). The default is SOSI=NO.

When you use the DFHWBOUT macro, be aware that the HTML text which you insert may affect the page layout generated from the BMS map fields. You may need to adjust the inserted text to ensure a correct page layout.

Customization examples

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The following sample shows a customizing macro definition. The first invocation of DFHMDX sets defaults for the values to be applied to subsequent invocations of DFHMDX by specifying * for the map set name and map name. Later invocations override or add to the parameters for specific maps in the map set. The continuation characters are in column 72, and the continued text is resumed in column 16.

About this task

```
MACRO
DFHMSX
DFHMDX MAPSET=*,MAP=*,                                          *
      PF1='Help',PF3='Exit',PF4='Save',PF9='Messages'
DFHMDX MAPSET=DFHWB0,MAP=*,                                     *
```

```
      TITLE='CICS Web Interface',                          *
      PF3='Messages'
DFHMDX MAPSET=DFHWB0,MAP=DFHWB02,                          *
      TITLE='CICS Web Interface Enable',                   *
      PF3='Save'
MEND
```

When CICS creates the templates for each of your BMS map definitions, it invokes the customizing macro specified on the SYSPARM parameter in the DFHMAPT procedure. If the SYSPARM parameter does not specify a customizing macro name, DFHMSX is used. Each macro is processed in sequence, and if applicable, the parameter values are stored. Where a duplicate parameter is specified for a particular map or map set, the new value replaces the previous value for that map or map set only.

- The first DFHMDX macro in this example specifies `MAPSET=*,MAP=*` and `PF3='Exit'`. This value of PF3 applies to every map set and map for which a different value is not specified in a subsequent DFHMDX macro.
- The second DFHMDX macro specifies `MAPSET=DFHWB0,MAP=*` and `PF3='Messages'`. This value of PF3 applies to every map in map set DFHWB0 for which a different value is not specified in a subsequent DFHMDX macro.
- The third DFHMDX macro specifies `MAPSET=DFHWB0,MAP=DFHWB02` and `PF3='Save'`. This value applies only to map DFHWB02 in map set DFHWB0.

The default template generated from the BMS map contains buttons to represent all the following keys:

- ENTER and CLEAR keys
- PA1, PA2, and PA3 keys
- Program function keys PF1 to PF24
- HTML reset

However, if you use the DFHMDX macro to specify the buttons you want in your template, only the buttons you specify will be included in the template. For example, if you specify

```
DFHMDX MAPSET=*,MAP=*,PF3='Exit',ENTER='Continue'
```

the template will contain buttons for the PF3 and ENTER keys only.

Here are further examples showing how you can customize the HTML template generated from a BMS map.

- **Support the application's use of keys that are not in the standard output.**

  You can add a button to the map AD001 as follows:

  ```
  DFHMDX MAP=AD001,PFxx='Resubmit'
  ```

  where PFxx is the new PF key number you want to specify. The Web client displays a button with the legend "Resubmit". If the user clicks this button, it is reported to the application as PFxx.

- **Suppress the HTML Reset function.**

  You can suppress the Reset function for the map AD001 as follows:

  ```
  DFHMDX MAP=AD001,RESET=NO
  ```

  The Web client displays a page that does not contain a Reset button.

- **Change the appearance of the buttons, or the text associated with them.**

  You can change the legend on the PF1 button as follows:

  ```
  DFHMDX PF1='Help'
  ```

The Web client displays a button with the legend "Help". If the user clicks this button, it is presented to the application as PF1.

- **Provide an HTML title for the HTML page.**

  You can add a title to a displayed map as follows:

  ```
  DFHMDX MAP=DFHWB01,TITLE='CICS Web Interface'
  ```

  The Web client displays "CICS Web Interface" as the title of the page.

- **Provide a masthead graphic for the HTML page.**

  Write a DFHMDX macro for the map that is to have the masthead. For example:

  ```
  DFHMDX MASTHEAD=(/dfhwbimg/masthead.gif,'CWI')
  ```

  The Web client uses the specified masthead, or will show "CWI" as the masthead if it cannot find the graphic file.

- **Change the color of the background, or specify a special background.**

  Write a DFHMDX macro for the map that is to have a special background. For example:

  ```
  DFHMDX MAP=AD001,BACKGROUND=/dfhwbimg/texture4.jpeg
  ```

  The Web client uses the specified file as a background for the page.

  To change the color of the background, use the BGCOLOR parameter.

- **Modify the BMS colors.**

  To modify the BMS colors, write a DFHMDX macro like the following:

  ```
  DFHMDX MAP=AD001,BLUE=AQUA,YELLOW=#FF8000
  ```

  The Web client shows BMS blue text in HTML aqua (the same as BMS turquoise), and BMS yellow text in bright orange.

- **Suppress parts of the BMS map.**

  You can suppress a field in a map as follows:

  ```
  DFHMDX MAP=AD001,SUPPRESS=((5,2),(6,2),(7,*))
  ```

  The displayed page does not contain the field at row 5 column 2, nor the field at row 6 column 2, nor any of the fields in row 7 of the map.

- **Add Web client control functions.**

  If you want a JavaScript function to be invoked when a page is loaded, use the ONLOAD parameter of the DFHMDX macro in your customization macro. For example, if you code:

  ```
  DFHMDX MAP=AD001,ONLOAD='jset(''CWI is wonderful'',''Hello there!'')'
  ```

  JavaScript function `jset()` is invoked with the given parameters when the page is loaded.

  To complete this customization, the definition of the `jset` function must be added to the header of the HTML page with a DFHWBOUT macro. You must put the macro invocation before the first DFHMDF macro in the BMS map definition. Here is a sample:

  ```
  DFHWBOUT '<script language="JavaScript">'
  DFHWBOUT 'function jset(msg,wng)'
  DFHWBOUT '         {window.status = msg; alert(wng)}'
  DFHWBOUT '</script>'
  ```

  When the page is loaded the status area at the bottom of the window contains the message "CWI is wonderful", and an alert window opens that contains the message "Hello there!".

- **Add text that appears only on the HTML page, but is not part of the BMS map.**

  Put DFHWBOUT macros in the BMS map definition at the point where you want the text to appear. For example:

```
DFHWBOUT '<p>This text illustrates the use of the DFHWBOUT macro,'
DFHWBOUT 'which can be used to output text that should only appear'
DFHWBOUT 'in HTML templates, and will never appear in the'
DFHWBOUT 'corresponding BMS map.'
```

  will produce the following lines in the HTML template:

```
<p>This text illustrates the use of the DFHWBOUT macro,
which can be used to output text that should only appear
in HTML templates, and will never appear in the
corresponding BMS map.
```

- **Add HTML header information to the HTML page.**

  Put DFHWBOUT macros in the BMS map definition before the first occurrence of DFHMDF. For example:

```
DFHWBOUT '<meta name="author" content="E Phillips Oppenheim">'
 DFHWBOUT '<meta name="owner"  content="epoppenh@xxxxxxx.yyy.co*
       m">'
 DFHWBOUT '<meta name="review" content="19980101">'
 DFHWBOUT '<meta http-equiv="Last-Modified" content="&WBDATE&W*
       BTIME GMT">'
```

  will produce the following lines in the head section of the HTML template:

```
<meta name="author" content="E Phillips Oppenheim">
<meta name="owner"  content="epoppenh@xxxxxxx.yyy.com">
<meta name="review" content="19980101">
<meta http-equiv="Last-Modified" content="23-Dec-1997 12:06:46 GMT">
```

  DFHMSD sets the values of &WBDATE and &WBTIME to the time and date at which the macro is assembled.

- **Use country-specific characters in JavaScript and HTML.**

  The default US code page 37, which is used to produce the template, can be modified for different code pages. For example:

```
DFHMDX OPENSQ=[,CLOSESQ=],OPENBR={,CLOSEBR=},EXCLAM=!
```

  This specifies the substitutions needed. The characters must be entered on a terminal where the code page corresponds to the SERVERCP on the DFHCNV call.

- **Make fields right-aligned in the HTML page.**

  You can right-align the data in a field as follows:

```
DFHMDX MAPSET=MAPSETA,MAP=AD001,RALIGN=((3,5),(*,15),(*,3),(6,7),(*,83))
```

  In this example, data will be right aligned in all the following fields

```
DFHMDF POS=(3,5),LENGTH=4,INITIAL='TEXT',ATTRB=PROT
DFHMDF POS=(5,80),LENGTH=3,INITIAL='123',ATTRB=PROT
DFHMDF POS=(2,10),LENGTH=5,INITIAL='  EXT',ATTRB=ASKIP
DFHMDF POS=(4,8),LENGTH=7,INITIAL='INITEX ',ATTRB=PROT
DFHMDF POS=(1,1),LENGTH=2,XINIT='C1C2',ATTRB=ASKIP
DFHMDF POS=(6,7),LENGTH=4,XINIT='0E44850F',ATTRB=PROT,SOSI=YES
DFHMDF POS=(2,9),LENGTH=6,XINIT='0E448544830F',SOSI=YES,ATTRB=PROT
DFHMDF POS=(2,9),LENGTH=6,XINIT='448544834040',PS=8,ATTRB=PROT
```

- **Make numeric fields right aligned**

  You can make all fields with the NUMERIC attribute right aligned within their html table cells as follows:

```
DFHMDX MAPSET=MAPSETA,MAP=AD001,NUMALIGN=YES
```

Chapter 17. CICS Web support in a CICSPlex

You can distribute applications that use CICS Web support in a CICSPlex using the following methods individually, or in combination:

- You can use network load balancing to distribute requests from Web clients to several CICS regions.
- CICS Web support and the business application can execute in the same CICS region.
- CICS Web support can execute in a router region, and the business application can execute in one or more application-owning regions (AORs). However, you cannot use the **EXEC CICS** WEB API in the AOR, so a Web-aware application program cannot execute in the AOR. You can use the **EXEC CICS** DOCUMENT API in the AOR, but you must provide your own mechanism for transferring the HTML output back to the router region. See "Routing a Web client's request to an AOR" on page 230 for more information.

Figure 11 illustrates these configurations.

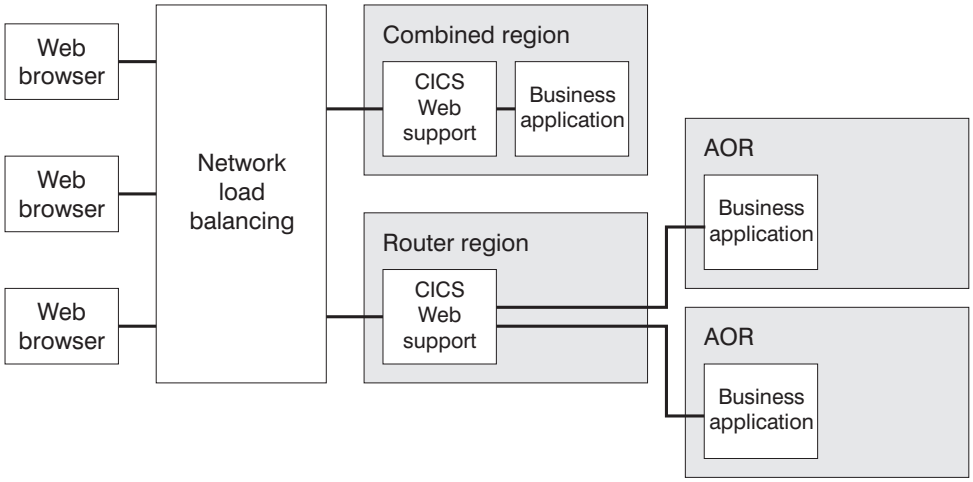


Figure 11. CICS Web support configurations in a CICSPlex

You can distribute requests that use the CICS business logic interface in the same way. This is illustrated in Figure 12 on page 230.

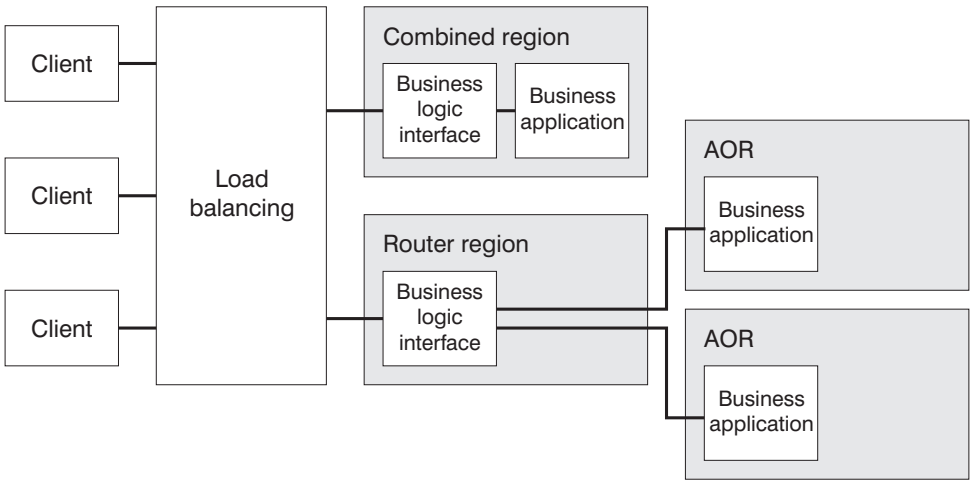


Figure 12. CICS business logic interface configurations in a CICSPlex

When you plan to distribute applications in this way, you should consider any affinities that exist between the parts of your application. For more information about affinities, see the *CICS Interdependency Analyzer for z/OS User's Guide and Reference*.

You should also consider how the application's state will be managed between requests. "Managing application state across an HTTP request sequence" on page 91 discusses the considerations involved for any CICS Web support applications which use a pseudoconversational model. There may be additional considerations when:

- Dynamic routing is used to select the AOR within which the business application executes.
- Workload and connection balancing is used to select the router region (and, indirectly, the AOR).

CICS provides a sample state management program (DFH$WBSR) that you can use to manage your application state. DFH$WBSR facilitates the sharing of application state through resources that can be shared by several CICS regions. It is described in Appendix J, "Reference information for DFH$WBST and DFH$WBSR, state management samples," on page 431. (The other sample, DFH$WBST, creates an affinity, and so is not suitable for use in a CICSPlex.)

For guidance about configuring CICS Web support and the CICS business logic interface in a CICSPlex, see *Workload Management for Web Access to CICS*.

Routing a Web client's request to an AOR

You cannot use the **EXEC CICS WEB** API in an AOR; you can use it only in the router region.

About this task

If you want to use a Web-aware application to respond to requests, one solution is to code your presentation logic in the Web-aware application program (which executes in the router region), and code your business logic (which executes in the AOR) to be entirely independent of presentation. The Web-aware application program is named as the program that handles the request, and it must manage its own communication with the application program that carries out the business

logic. “HTTP request and response processing for CICS as an HTTP server” on page 26 explains the processing stages for Web-aware applications.

For non-Web-aware applications, a converter program in the router region can be used to produce an HTTP response from information supplied by an application program in an AOR. Figure 13 shows the processing stages and task structure associated with a request from a Web client when a non-Web-aware application program is executed in an application-owning region.

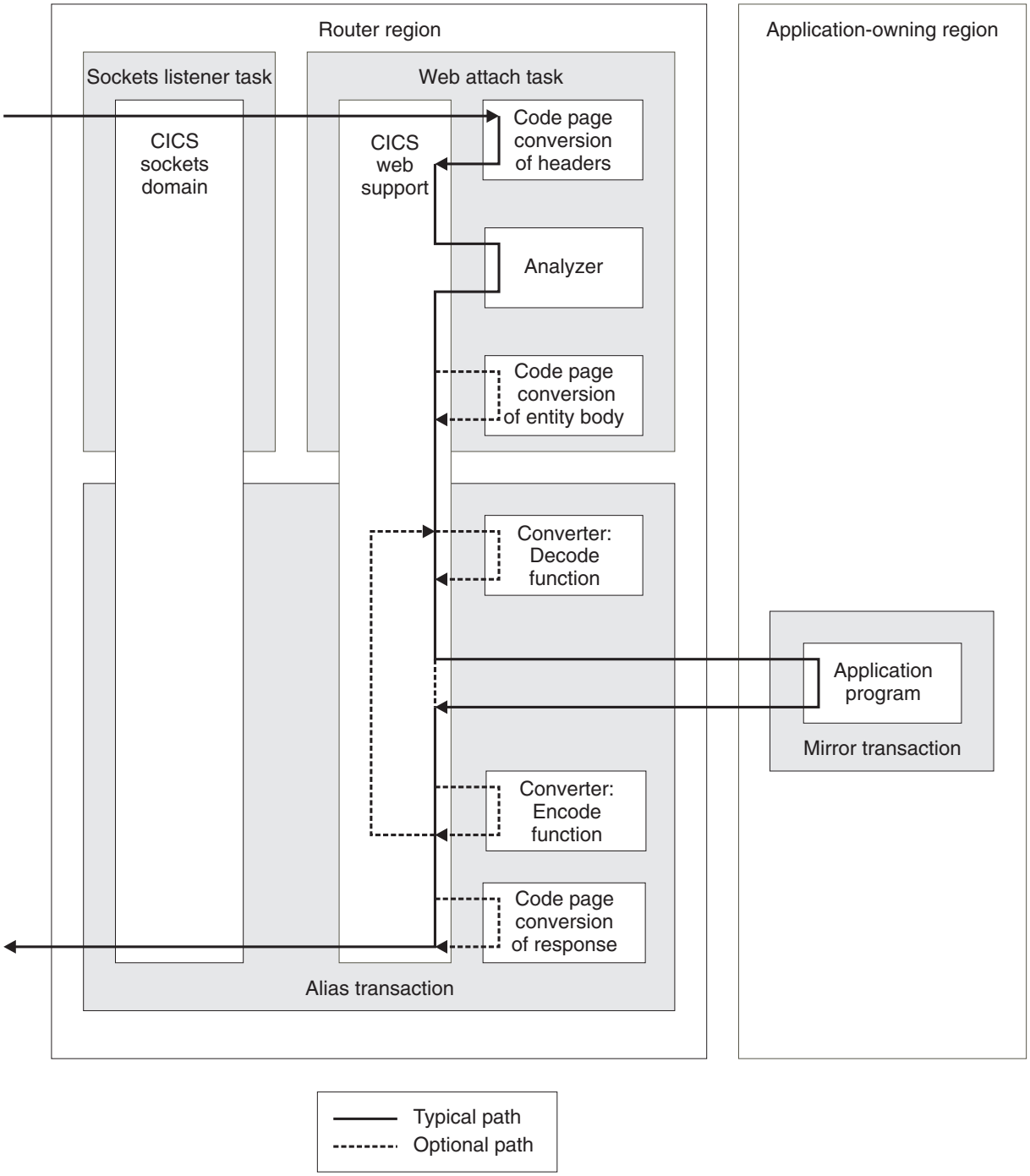


Figure 13. How CICS Web support routes non-Web-aware application requests to an AOR

The corresponding stages for the CICS business logic interface are shown in Figure 14.

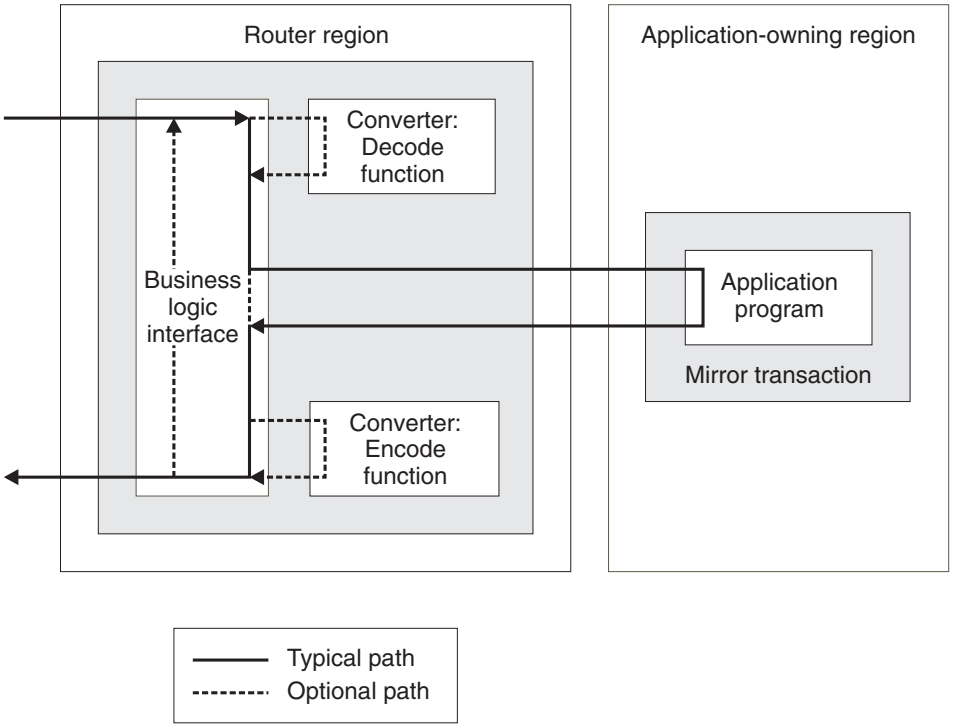


Figure 14. How the CICS business logic interface routes application requests to an AOR

CICS uses Distributed Program Link (DPL) to invoke the application program in the AOR; the application executes under the mirror task. For information about DPL, see the *CICS Intercommunication Guide*.

To execute a business application in an AOR:

- You must specify the REMOTESYSTEM attribute, or specify DYNAMIC(YES) on the PROGRAM definition for the application program. If you specify DYNAMIC(YES), the dynamic routing program determines where the application program executes.
- Other resource definitions (for the analyzer program, the Web-aware application program or converter program, and the alias transaction) must specify that they execute in the router region.
- You must define an MRO or APPC connection between the router region and the AOR.

If the application program which executes in the AOR is designed to be entirely independent of presentation, it returns output to the Web-aware application program or the converter program, which then constructs the HTML output. Alternatively, if you are using a converter program, you might want to use the EXEC CICS DOCUMENT API in the AOR to construct HTML output. The converter program can use this output to produce a complete HTTP response.

You must provide your own mechanism for transferring the application program's output back to the router region. The output can be transferred in a COMMAREA. Alternatively, you can use some other mechanism, such as a temporary storage queue, and transfer a token representing the data in that mechanism. The program in the router region can use the token to retrieve the output, then process it and

pass it to the Web client. CICS provides a sample state management program (DFH$WBSR) that you can use to do this. It is described in Appendix J, "Reference information for DFH$WBST and DFH$WBSR, state management samples," on page 431 (the other sample, DFH$WBST, creates an affinity, and so is not suitable for use in a CICSPlex).

Network load balancing

To avoid being dependent on a single CICS router region, consider using more than one router region to share the workload from the network.

About this task

There are several techniques that you can use to balance the workload between your router regions:

Sysplex Distributor
Sysplex Distributor is a feature of z/OS Communications Server that uses a cluster IP address that is a dynamic application-specific virtual IP address (VIPA), that represents a cluster of CICS server instances in the sysplex. For more information about Sysplex Distributor, see *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

Application-specific virtual IP address (VIPA)
An application-specific VIPA is a feature of z/OS Communications server that provides a bind-activated virtual IP address. The VIPA becomes active on a TCP/IP stack when a CICS region is started, and the VIPA can move around a sysplex with a specific CICS region. For more information about VIPA, see *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

DNS approach
DNS connection optimization balances IP connections in an z/OS sysplex IP domain, based on feedback from MVS™ WLM about the health of the registered applications. It is still supported for CICS use. For information about DNS/WLM support, see *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

Port sharing
TCP/IP port sharing provides a simple way of spreading HTTP requests over a group of CICS router regions running in the same z/OS image. CICS TCPIPSERVICEs in different regions are configured to listen on the same port, and TCP/IP is configured with the SHAREPORT option. The TCP/IP stack then balances connection requests across the listeners. For more information about TCP/IP port sharing, see *z/OS Communications Server: IP Configuration Reference*, SG24-5466.

Part 3. Atom feeds from CICS - start here

CICS can serve Atom feeds to Web clients. The Atom feeds consist of data that is supplied by CICS resources or application programs. When you expose a CICS resource or application program as an Atom feed or collection, users can read and update the data by making HTTP requests from external client applications, such as feed readers or Web mashup applications.

CICS provides samples that you can use to help you set up an Atom feed or collection using your own data. To try out CICS support for Atom feeds, follow the Web 2.0 scenarios.

Chapter 18. Atom feeds

A Web feed, sometimes just called a "feed", is a series of related items that a content provider publishes on the Internet. An Atom feed is a Web feed that uses the Atom Syndication Format and the Atom Publishing Protocol.

Atom comprises an XML-based format that describes an Atom feed and the items of information in it, and a protocol for publishing and editing Atom feeds. This format and protocol are described in two Internet Society and IETF (Internet Engineering Task Force) Request for Comments documents (known as RFCs):

RFC 4287, *The Atom Syndication Format*, available from http://www.ietf.org/rfc/rfc4287.txt

RFC 5023, *The Atom Publishing Protocol*, available from http://www.ietf.org/rfc/rfc5023.txt

Content providers often deliver Web feeds in an earlier format called RSS (Really Simple Syndication). CICS supports Atom, but does not support RSS.

The items of information that make up an Atom feed are known as *Atom entries*. A content provider publishes, or "syndicates", an Atom feed by making it available through a URL on the Internet and updating it with new items. Web pages can display the items in the Atom feed, and Web users can obtain the items from the feed using a feed reader or Web browser. An Atom feed might be used as part of a *mashup*, which is a Web application that merges content from a number of data sources so that users can experience and understand the data in a new way. In a mashup, the data from the Atom feed can be handled by a *widget*, which is a script application that runs in a Web page.

The Atom Publishing Protocol specifies the way that users can add, delete, edit, or view individual Atom entries in an Atom feed by making HTTP requests to a server that stores the entries. A GET request retrieves an entry for viewing, a POST request adds a complete new entry, a PUT request edits an existing entry, and a DELETE request deletes an entry. The server handles the requested changes in an appropriate way and responds to the user's client with confirmation of the changes.

Atom entry documents

An Atom entry document is an XML document that contains a single item of information, known as an entry, for the Atom feed.

An Atom entry document consists of an <atom:entry> element that contains a number of child elements. The child elements provide the content for the entry and also metadata about the entry, such as its title or the time when it was first published.

The content of an Atom entry can be plain text, HTML, XHTML, or another IANA (Internet Assigned Numbers Authority) media type. IANA media types are listed at http://www.iana.org/assignments/media-types/. An Atom entry can also have as its content a link to a media resource such as a movie or image, in which case it is called a media link entry.

The media type for an Atom entry document is application/atom+xml.

The format of Atom entry documents is defined by RFC 4287, *The Atom Syndication Format*, which is available from http://www.ietf.org/rfc/rfc4287.txt.

Atom feed documents

An Atom feed document is an XML document that provides metadata about an Atom feed and one or more entries for the feed. When a client makes a request for information from the feed, the server generates a feed document that includes a suitable number of Atom entries to fulfil the request.

An Atom feed document consists of an <atom:feed> element that contains a number of child elements. The <atom:entry> element is the most important child element, but normally the entries for the feed exist as separate XML documents, and the server adds them when it serves the feed document. An Atom feed document is still an acceptable document when it does not contain any <atom:entry> elements.

The other child elements contain metadata about the feed, such as its title and subtitle, or its main author. Some of the items of metadata in the Atom feed document, such as the author's details and the information about intellectual property rights, can apply to all the entries in the feed unless an entry includes its own version of that item of metadata.

The media type for an Atom feed document is application/atom+xml.

The format of Atom feed documents is defined by RFC 4287, *The Atom Syndication Format*, which is available from http://www.ietf.org/rfc/rfc4287.txt.

Atom collections

An Atom collection is a special kind of Atom feed document that lists the URLs of Atom entries that are available to be edited. Its format is like that of an ordinary Atom feed document with the addition of some specialized elements. It is distinguished as a collection by being listed in an Atom service document.

An Atom collection contains some specialized <atom:link> elements. If the collection is large enough that more than one feed document is required to return all the entries, the elements <atom:link rel="first">, <atom:link rel="last">, <atom:link rel="next">, and <atom:link rel="previous"> supply navigation between the feed documents. Entries also have an <atom:link rel="edit"> link to which edit requests can be directed. The <app:edited> element is added to entries in a collection to state the time of the last edit for each entry.

When Atom entries are made available as a collection, a client can edit or delete the existing entries and create new entries for the collection. The client manipulates the entries by sending HTTP requests to the server as follows:

GET Retrieve a single Atom entry or a list of Atom entries. GET requests for a list of Atom entries are sent to the URL of the collection, as stated in the Atom service document. GET requests for a single Atom entry are sent to the URL of an individual Atom entry in the collection, as stated in the <atom:link rel="edit"> link for the entry.

POST Create a new Atom entry. POST requests are sent to the URL of the collection.

PUT Edit an existing Atom entry that the client has obtained using a GET request. PUT requests are sent to the URL of an individual Atom entry in the collection.

DELETE
Delete an existing Atom entry. DELETE requests are sent to the URL of an individual Atom entry in the collection.

The server sends an appropriate HTTP response to the client in each case. A server can change the metadata that the client provides for the entry, so when a client makes a successful POST or PUT request, the server also returns a copy of the new entry as the body of the response.

As well as containing standard Atom entries, a collection can also contain media resources, such as a movie or image. If a server supports media resources, it creates special Atom entries known as media link entries in the collection to provide links to these resources. CICS does not provide support for media resources.

The format of Atom collections is defined by RFC 5023, *The Atom Publishing Protocol*, which is available from http://www.ietf.org/rfc/rfc5023.txt.

Related concepts:

"URLs for Atom feeds from CICS" on page 245
Atom feed documents, collections, and Atom entry documents within feeds or collections, contain URLs (Uniform Resource Locators) that Web clients can use to interact with the documents. Each URL is provided in an <atom:link> element in the Atom document. An Atom document can have more than one <atom:link> element, and the rel attribute of the element, known as the link relation, specifies the purpose of the different URLs.

"Internationalized Resource Identifiers (IRIs)" on page 249
Internationalized Resource Identifiers (IRIs) are a form of resource identifier for the Internet that permits the use of characters and formats that are suitable for national languages other than English. IRIs can be used in place of URIs or URLs where the applications involved with the request and response support them.

Atom service documents

An Atom service document is an XML document that lists the collections that are available from a server.

An Atom service document has the root element <app:service>. (The app: prefix is the namespace prefix for the Atom Publishing Protocol.) It has one or more <app:workspace> elements that define workspaces containing a number of <app:collection> elements. A workspace is used only for grouping collections; you cannot perform any actions on a workspace.

The <app:collection> elements list the URL and title of each collection, and might also state the types of input that the collection accepts and the categories that can be used for entries.

The media type for an Atom service document is application/atomsvc+xml.

The format of Atom service documents is defined by RFC 5023, *The Atom Publishing Protocol*, which is available from http://www.ietf.org/rfc/rfc5023.txt.

Atom category documents

A category document contains lists of categories for the entries in a collection. Categories can also be specified in a service document. Separate category documents are useful if you want to use the same categories to define multiple Atom feeds.

An Atom category document has the root element <app:categories>. (The app: prefix is the namespace prefix for the Atom Publishing Protocol.) The <app:categories> element contains a list of <atom:category> elements that are permitted for entries in a collection. The list of categories can be fixed, in which case the server can reject entries with other categories, or it can be open, so that other categories can be used.

If a separate Atom category document is used in the place of a list of categories in an Atom service document, the category document is referenced in the service document by its URL.

The media type for an Atom collection is application/atomcat+xml.

The format of Atom category documents is defined by RFC 5023, *The Atom Publishing Protocol*, which is available from http://www.ietf.org/rfc/rfc5023.txt.

Chapter 19. How CICS supports Atom feeds

CICS supports Atom feeds using the HTTP server functions of CICS Web support, and some additional functions to carry out the actions required of a server that supports the Atom format and protocol. You must select or set up a resource that provides the data for your Atom feed, and define the feed to CICS.

Before serving an Atom feed from CICS, you must configure the base components of CICS Web support to set CICS up as an HTTP server.

You can create Atom feeds from data held in or produced by existing resources, such as a temporary storage queue, a file, records in a database application, a Web service, or output produced by an existing application program. A single record in the resource holds the data for a single Atom entry. Alternatively, you can set up a new resource to contain Atom entries.

If your resource is a file or temporary storage queue defined to CICS, and you have a language structure written in COBOL, C, C++, or PL/I that describes the records in the resource, CICS can extract data directly from the resource to produce the Atom feed. You use the language structure as input to the CICS XML assistant program to produce an XML binding that defines the structure of the resource, so that CICS can map the data to the correct elements in the Atom document.

You can also serve any resource as an Atom feed by writing a program, known as a service routine, that extracts data from each record in the resource to form an Atom entry, and supplies the data to CICS in a set of containers. If you are able to produce an XML binding for your resource, the service routine can make use of the information in the XML binding, but the service routine does not require an XML binding.

When you have identified or created the resource, and produced an XML binding or written a service routine, you define the Atom feed to CICS by creating the following items:

- An ATOMSERVICE resource definition to specify where CICS obtains the data to produce Atom documents in response to a Web client request.
- A URIMAP resource definition to specify how CICS handles HTTP requests from Web clients for the Atom feed. The URIMAP resource references the ATOMSERVICE resource definition. To support your URIMAP resource definition, you must have a TCPIPSERVICE definition that defines an inbound port for CICS Web support, on which CICS can receive HTTP requests.
- An Atom configuration file, which contains the XML syntax for the Atom feed document, together with some elements specific to CICS, such as elements to identify the resource that contains the data for the feed. CICS uses the information in the Atom configuration file to construct an Atom feed document containing a number of Atom entries, which CICS produces using the data from your resource.

If you want to enable Web clients to manage and edit the Atom entries in the feed, you can take further steps to set up the Atom feed as a collection. To set up a collection, you create a new URIMAP definition to make the collection available separately from the feed. You also create a new ATOMSERVICE definition and Atom configuration file by copying the equivalent files for the Atom feed from the

same data, redefining them to state that they are for a collection, and making minor changes. You then create an Atom service document and optionally an Atom category document to define your collection, and make those documents available through CICS. If you are using a service routine, you must code it to handle Web client requests to add, edit, and delete Atom entries from the collection.

Interacting with Atom feeds

When you have set up an Atom feed, Web clients can access it to obtain a list of Atom entries. CICS, together with your service routine if used, acts as a server to receive the Web clients' HTTP requests and return Atom feed documents containing a number of Atom entries. Many free or commercially available Web client applications are able to request, receive and display Atom feeds, including most modern Web browsers, dedicated feed readers, and applications that provide further functions, such as applications for creating mashups. Check that the application is described as supporting the Atom format. You can also write your own Web client application to make GET requests for Atom feed data.

If you have also set up your Atom feed as a collection, you or others can manage and edit the entries in the feed through a Web client that supports HTTP POST, PUT, and DELETE requests for Atom feeds, as described in the Atom Publishing Protocol. If you do not have a Web client with this capability, you can use a Web client application that lets you compose and send your own HTTP requests and view the responses. You can also write your own Web client applications to make POST, PUT, and DELETE requests to Atom collections. If CICS is managing your resource directly, CICS applies the Web clients' editing requests to the data that you have made available in the collection, and returns an appropriate response. If you are using a service routine to provide your data, CICS passes the Web clients' requests to the service routine using the container interface, and you code the service routine to modify the resource in response to the requests.

To learn more about CICS support for Atom feeds and how Web clients interact with them, follow the Web 2.0 scenarios.

Atom feeds from the CA8K SupportPac

If you used the CA8K SupportPac in CICS TS for z/OS, Version 3.1 or CICS TS for z/OS, Version 3.2 to set up Atom feeds, and you want to upgrade to use the support for Atom feeds in CICS TS for z/OS, Version 4.1, you can continue to use your service routines. However, instead of PIPELINE resource definitions, pipeline configuration files, and Resource Layout Mapping structures, you must use ATOMSERVICE resource definitions, Atom configuration files, and XML bindings. You must also make changes to your service routine code to rename the containers and to account for new parameters in one of the containers, then recompile the modules.

Chapter 20. How Atom feeds work in CICS

To serve an Atom document, CICS must include suitable URLs, identify and obtain the data for the Atom entries, and determine the arrangement of the Atom entries in the Atom document. When you set up an Atom feed or collection, you must provide information in your Atom configuration file or your service routine to help CICS complete these tasks.

Related concepts:

Chapter 18, "Atom feeds," on page 237
A Web feed, sometimes just called a "feed", is a series of related items that a content provider publishes on the Internet. An Atom feed is a Web feed that uses the Atom Syndication Format and the Atom Publishing Protocol.

Chapter 19, "How CICS supports Atom feeds," on page 241
CICS supports Atom feeds using the HTTP server functions of CICS Web support, and some additional functions to carry out the actions required of a server that supports the Atom format and protocol. You must select or set up a resource that provides the data for your Atom feed, and define the feed to CICS.

Data processing for Atom feeds from CICS

To produce an Atom feed or collection document containing Atom entries, CICS obtains data for the Atom entries either directly from a file or temporary storage queue, or from a service routine that extracts data from another resource.

Figure 15 shows how CICS uses an Atom configuration file to identify and extract relevant data from a record in a file:

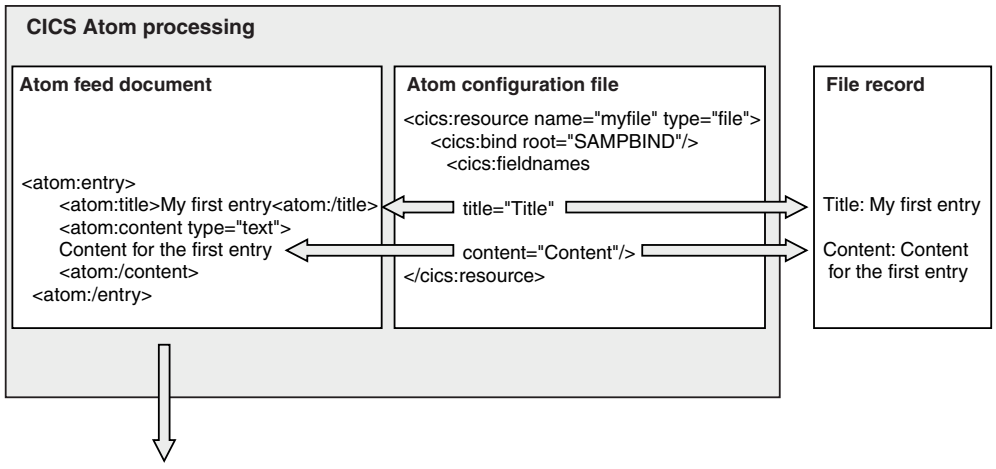


Figure 15. Extracting Atom entry data directly from a file

- The record in the file contains fields called "Title" and "Content" that hold data for the Atom entry.
- The Atom configuration file includes a <cics:resource> element that identifies the file, a <cics:bind> element that refers to the XML binding for the file, and a <cics:fieldnames> element. The title attribute of the <cics:fieldnames> element identifies the "Title" field in the file record as the field that holds the data for the

title of the Atom entry. The content attribute identifies the "Content" field in the file record as the field that holds the data for the content of the Atom entry.

- CICS uses the information in the Atom configuration file and the XML binding to locate and extract the data from the "Title" and "Content" fields in the file record, and uses that data to populate the <atom:title> and <atom:content> elements of an Atom entry in an Atom feed document.
- When CICS has carried out this processing for a series of records from the file to produce the required number of Atom entries, CICS sends the Atom feed document containing the Atom entries to the Web client.

Figure 16 shows how CICS obtains data from a record in a database through a user-written service routine program:

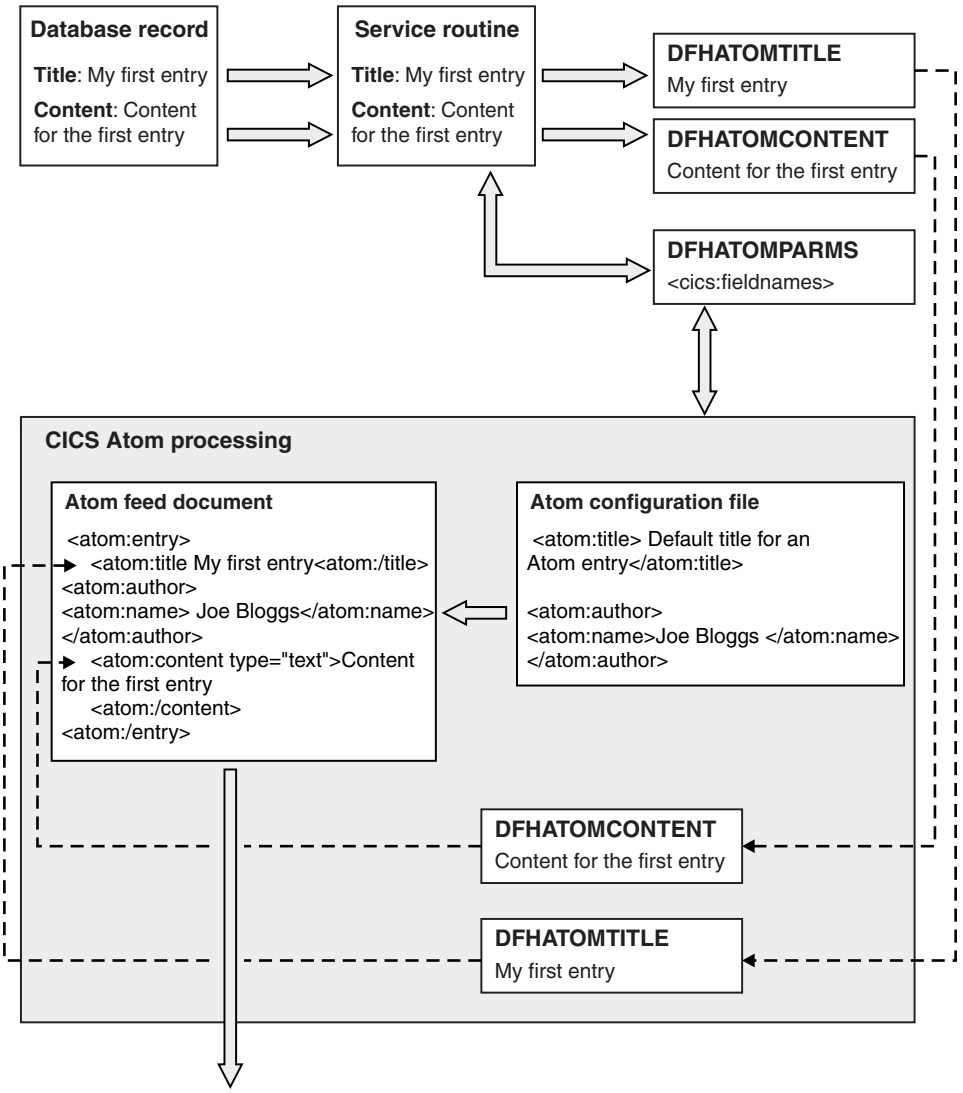


Figure 16. Using a service routine to supply Atom entry data

- The record in the database contains fields called "Title" and "Content" that hold data for the Atom entry.
- A service routine extracts the data from the "Title" and "Content" fields in the database. If a service routine is working with a resource that has an XML binding, it can obtain the names of the relevant fields from information in a

<cics:fieldnames> element in an Atom configuration file, which CICS passes to the service routine as parameters in a container called DFHATOMPARMS. DFHATOMPARMS also contains other information about the Web client's request.

- The service routine creates containers called DFHATOMTITLE and DFHATOMCONTENT, and writes the data from the "Title" and "Content" fields into the containers. It then returns the containers to CICS Atom processing.
- The Atom configuration file includes an <atom:title> element that gives a default title for Atom entries, and an <atom:author> element that contains an <atom:name> element giving the author name Joe Bloggs.
- CICS composes an Atom entry using the title and content that were supplied by the service routine in the DFHATOMTITLE and DFHATOMCONTENT containers. The service routine did not supply an author name, so CICS uses the author name from the Atom configuration file. CICS does not need the default title from the Atom configuration file, because the service routine has supplied that data.
- When CICS has called the service routine a number of times to supply data from different database records to produce the required number of Atom entries, CICS sends the Atom feed document containing the Atom entries to the Web client.

URLs for Atom feeds from CICS

Atom feed documents, collections, and Atom entry documents within feeds or collections, contain URLs (Uniform Resource Locators) that Web clients can use to interact with the documents. Each URL is provided in an <atom:link> element in the Atom document. An Atom document can have more than one <atom:link> element, and the rel attribute of the element, known as the link relation, specifies the purpose of the different URLs.

An Atom feed document or collection served by CICS contains up to four types of URL:

- A URL that locates the whole of the Atom feed or collection. This feed URL is provided in an <atom:link rel="self" > element that is a child element of the <atom:feed> element. A Web client can use this URL to obtain an Atom feed document containing multiple entries from the Atom feed or collection.
- Individual URLs to locate each Atom entry in the feed or collection. These entry URLs are provided in <atom:link rel="self" > elements that are child elements of the <atom:entry> element. A Web client can use these URLs to retrieve single Atom entries from the feed or collection.
- Editing URLs that Web clients can use to make requests to edit a collection. These URLs are provided in <atom:link rel="edit" > elements. CICS provides one editing URL for the whole of a collection, as a child element of the <atom:feed> element in the collection document, and individual editing URLs for each Atom entry in a collection, as child elements of the <atom:entry> elements. CICS also provides <atom:link rel="self" > URLs for collections and Atom entries in collections.
- Navigation URLs that Web clients can use to retrieve partial lists of the Atom entries in an Atom feed or collection. These URLs are provided in <atom:link> elements with rel attributes of "first", "previous", "next", and "last". These URLs enable Web clients to explore the whole of an Atom feed or collection without having to retrieve all the Atom entries at once. CICS provides an <atom:link rel="next"> element in Atom feed documents with a URL that Web clients can use to retrieve the next window of Atom entries from the feed. In Atom documents that contain partial lists of entries from collections, CICS adds

<atom:link> elements with rel attributes of "first", "previous", "next", and "last", to provide navigation to the other partial lists of Atom entries from the collection.

For an Atom feed, the URL for the whole feed is typically publicized on the Internet or a company's intranet. When a Web client obtains an Atom feed document by using the feed URL, the Atom entries in the Atom feed document include their own individual URLs, and a Web client can use these to retrieve single Atom entries.

For a collection, which contains Atom entries that can be edited, the service document that is available from the server provides the editing URL of each of the collections on the server. A Web client can use one of these URLs to view the Atom entries in the collection and make requests to add further entries to it. The Web client can use the editing URL for an individual Atom entry to make a request to update or delete the entry.

The Atom Syndication Format allows the use of Internationalized Resource Identifiers (IRIs), which permit Unicode characters and formats that are suitable for national languages other than English. You may use IRIs that include Unicode characters as the resource locators for Atom feeds from CICS, in place of an ordinary URL. In the RFCs for the Atom Syndication Format and the Atom Publishing Protocol, the resource locators for Atom feeds and Atom entries are referred to as IRIs. "Internationalized Resource Identifiers (IRIs)" on page 249 explains IRIs and how you can use them for Atom feeds.

How Atom URLs are specified and resolved

In CICS, you use the <atom:link> child elements of the <atom:feed> and <atom:entry> elements in an Atom configuration file to specify a URL for the whole of the Atom feed or collection, and also a standard URL for the individual Atom entries. In the Atom configuration file you always specify <atom:link rel="self"> for these child elements, and when CICS sends out the Atom document, CICS adds an identical link in an <atom:link rel="edit" > element to collections and Atom entries in collections. You may omit the scheme and host components of the URL from the Atom configuration file, and specify only the path component. CICS adds the scheme and host components to the URLs when it returns the feed or entry document to the client.

You do not need to specify any of the <atom:link> elements for navigation URLs, with rel attributes of "first", "previous", "next", and "last", in your Atom configuration file. CICS creates these links for you.

The URLs that you specify for the whole feed and as a standard URL for the individual entries must have path components that begin in the same way. You specify this common part of the path component in the URIMAP resource definition that CICS uses to handle Web client requests for the Atom feed, and use an asterisk to indicate that the rest of the path is to be used for path matching. The common part of the path component is what CICS uses to identify the Atom feed or collection, so it must be unique to this Atom feed or collection among all the Atom feeds or collections that you serve using a given host name.

When a Web client makes a request using a URL that includes this common part of the path component, CICS finds the matching URIMAP resource definition, and uses a number of other resources to map the request URL to the data for the Atom

feed. Figure 17 shows this process for a feed URL:

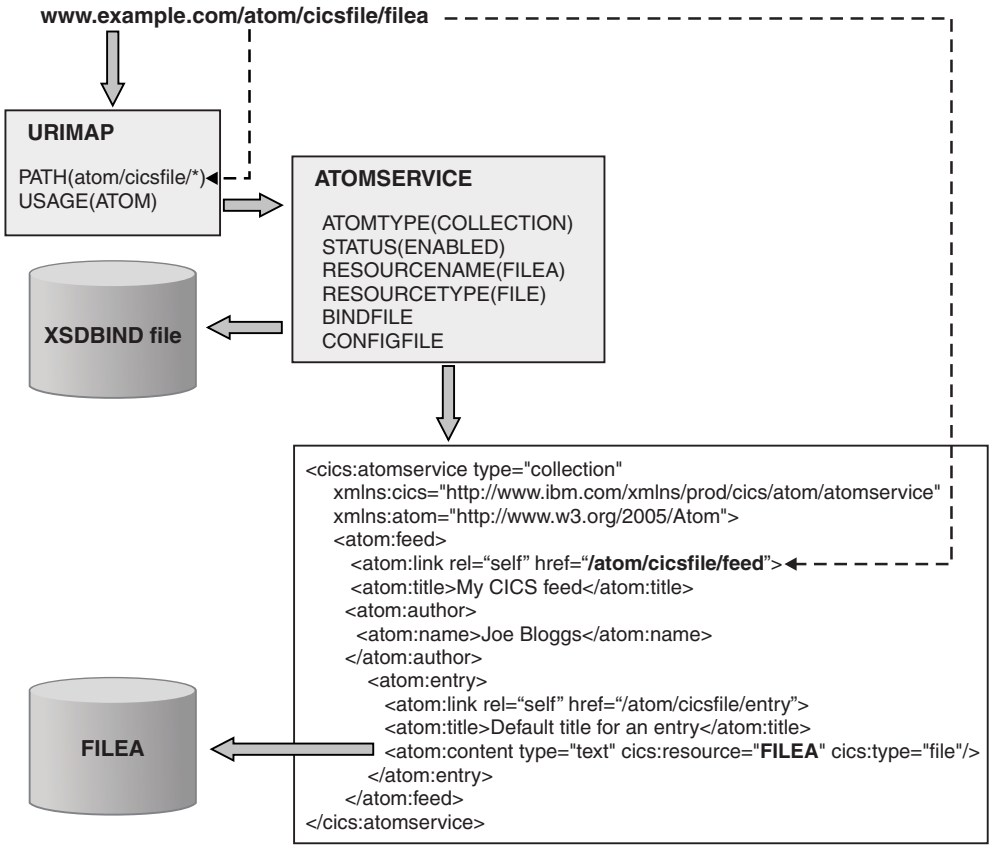


Figure 17. Request URLs for Atom feeds

- To handle incoming requests from Web clients, you create a URIMAP resource definition that specifies the part of the path component that is common to the feed and entry URLs. In this example, the common part of the path component is `atom/cicsfile/`. When a Web client makes a request using the URLs that you have defined for an Atom feed or collection or for an Atom entry, CICS finds the URIMAP resource that matches the common part of the path component. In this example, the Web client requests the Atom feed using the feed URL `www.example.com/atom/cicsfile/filea`.
- The URIMAP resource specifies an ATOMSERVICE resource that names the Atom configuration file, XML binding (XSDBIND file), and CICS resource that provide the Atom feed. The example ATOMSERVICE resource names the FILEA file as the resource that holds the data for the Atom entries.
- CICS uses the ATOMSERVICE resource to locate the Atom configuration file, and compares the path component of the inbound URL used by the Web client to the URLs specified in all the <atom:link> elements in the Atom configuration file. When CICS finds a URL in an <atom:link> element that has a matching path component, it carries out the appropriate action, either returning the Atom feed or entry document or implementing the edit request. In this example, the request URL used by the Web client matches the URL specified for the Atom feed in the Atom configuration file, so CICS must return an Atom feed document.
- The Atom configuration file, like the ATOMSERVICE resource, names the FILEA file as the resource that holds the data for the Atom entries. As explained in

"Data processing for Atom feeds from CICS" on page 243, CICS might operate directly to extract the data from the file or temporary storage queue that contains the data for the Atom entries, or pass the request on to a service routine.

In Figure 17 on page 247, the path for the URL for the whole Atom feed, as specified in the <atom:link> child element of the <atom:feed> element in the Atom configuration file, is `/atom/cicsfile/filea`. The <atom:entry> element in the Atom configuration file also has an <atom:link> child element, which contains the path `/atom/cicsfile/entry`. This is a standard path for Atom entries. The standard path for Atom entries begins with the common part of the path component, `atom/cicsfile/`. The remainder of the standard path for Atom entries must be different from the path for the Atom feed that is specified in the <atom:link> child element of the <atom:feed> element. CICS uses this part of the path for path matching within the Atom configuration file, to determine whether an Atom feed document or an Atom entry document is required.

Figure 18 shows how CICS handles a request from a Web client for a single Atom entry, and identifies the correct Atom entry:

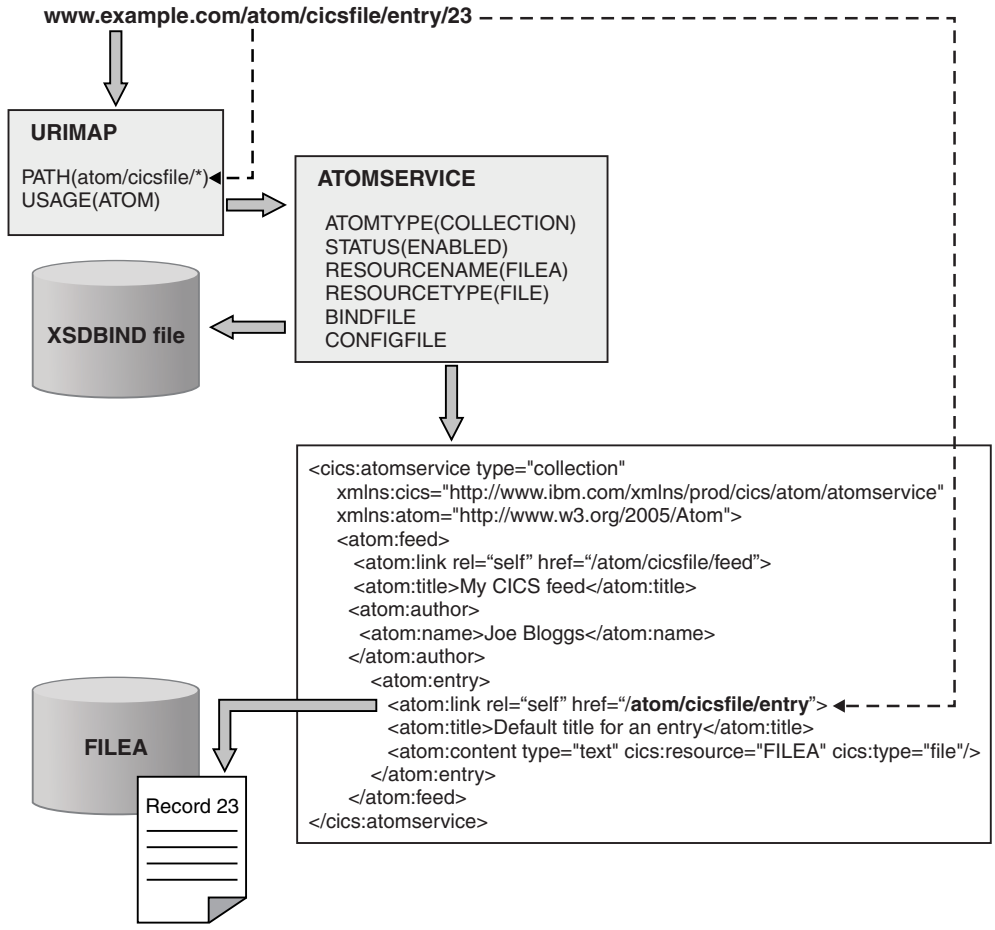


Figure 18. Request URLs for Atom entries

- The Web client requests a single Atom entry using the URL `www.example.com/atom/cicsfile/entry/23`. The Web client obtained this URL

from the <atom:link> child element for the Atom entry, which the Web client originally received as part of an Atom feed document.

- The Atom entry URL contains the common part of the path component for the Atom feed, `atom/cicsfile/`, so it is handled by the same URIMAP and ATOMSERVICE resources as the feed URL. In this example, the request URL used by the Web client matches the standard path specified for Atom entries in the Atom configuration file, so CICS must return an Atom entry document.
- CICS identifies the Atom entry to return to the Web client by examining the remainder of the request URL that follows the standard path. In this example, the request URL contains the number "23". This is the selector value for the entry. The selector value is an identifier, typically a number, that identifies the record in a file, temporary storage queue, or other resource that contains the data for the Atom entry. In this example, the selector value chosen for the Atom entries was the record number. When you choose a selector value, you must make sure that the URL for the whole Atom feed and the standard path for Atom entries will still be different when the selector value is appended to them. "Selector value for Atom entries" on page 251 explains in more detail how selector values are chosen and used.

CICS also uses the selector value to build navigation links to partial lists of entries from the Atom feed or collection, in the <atom:link> elements with rel attributes of "first", "previous", "next", and "last". CICS builds these navigation links by taking the path that you specified for the whole of the Atom feed or collection, and appending the selector value for the Atom entry that appears at the beginning of the partial list. CICS uses this information together with the window setting specified for the Atom feed or collection to return a partial list to the Web client. In the example Atom feed used here, for a partial list of entries that begins with the Atom entry with the selector value "9", CICS creates the link `www.example.com/atom/cicsfile/filea/9`.

These examples show the selector value being appended to the URLs in the default format, known as the "segment" format, where the selector value is placed as the final segment of the path component. As an alternative, you can choose a URL style that is compatible with applications developed using the CA8K SupportPac, where the selector value for the Atom entry is placed in a query string. You can specify this alternative "query" format using the <cics:selector> element in the Atom configuration file. If <cics:selector style="query"/> is specified for the example Atom feed used here, CICS creates the links for individual Atom entries in the format `www.example.com/atom/cicsfile/entry?s=23`. The same format is used for the navigation links.

Internationalized Resource Identifiers (IRIs)

Internationalized Resource Identifiers (IRIs) are a form of resource identifier for the Internet that permits the use of characters and formats that are suitable for national languages other than English. IRIs can be used in place of URIs or URLs where the applications involved with the request and response support them.

IRIs are described by RFC 3987, *Internationalized Resource Identifiers (IRIs)*, which is available from http://www.ietf.org/rfc/rfc3987.txt. CICS supports the use of IRIs in URIMAP resources for inbound Web client requests to CICS as an HTTP server, and in Atom feed documents.

Host name

To accommodate the requirements of domain name servers, Web clients convert the host name in an IRI into a format called Punycode. Punycode is described by RFC 3492, *Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*, which is available from http://www.ietf.org/rfc/rfc3492.txt. This algorithm encodes the hostname into a string composed only of alphanumerics, hyphens, and periods.

If you want to use an IRI as the link for a Web resource or Atom feed that is served by CICS, in the URIMAP resource definition that defines the Web client's request to CICS, you must specify the host name in Punycode. CICS does not provide a tool to carry out this conversion, but free applications are available on the Internet to support the conversion of Unicode to Punycode. If you use a single asterisk in place of the host name, to make the URIMAP resource match any host name, you do not need to use Punycode.

Path component

Web clients do not convert the path component of an IRI into Punycode, but they do escape, or percent-encode, Unicode characters in the path.

If you are using an IRI for a Web resource that is served by CICS, in the URIMAP resource definition, you must percent-encode any Unicode characters in the path that you specify. If you do not have an application that can convert Unicode characters to percent-encoded representations, free applications are available on the Internet to perform this task. Note that the limits on URL length listed in "URLs for CICS Web support" on page 33 apply also to URLs for Atom feeds, which means that the part of the path component of the URL that you specify in the URIMAP resource definition must be 255 characters or less. A character in this context means a single ASCII character, not the original Unicode character. For example, the Cyrillic character that has the percent-encoded representation `%D0%B4` counts as 6 characters from the 255–character limit.

When CICS installs the URIMAP resource definition, CICS stores the path in the canonical form recommended for URIs and unescapes some of the characters, but the path that is displayed when you view the URIMAP resource remains as you entered it.

When you use an IRI as a link for an Atom feed or entry document, you specify the IRI in the Atom configuration file as well as in the URIMAP resource definition. You must percent-encode any Unicode characters in the IRI in the Atom configuration file.

When CICS issues an Atom document containing the IRI, CICS converts the percent-encoded characters to XML character references, so that the XML is valid. To use the resulting link in a Web client request, you must convert the XML character references back into percent-encoded characters.

This example URIMAP resource contains a path that uses Unicode characters to specify the beginning of an IRI for an Atom feed, with an asterisk at the end to indicate that path matching is used for the remainder of the IRI:

```
Urimap         : ALEXANDR
Group          : IRIMAPS
DEScription    :
STatus         : Enabled           Enabled | Disabled
```

```
  USAge          : Atom              Server | Client | Pipeline | Atom
 UNIVERSAL RESOURCE IDENTIFIER
  SCheme         : HTTP              HTTP | HTTPS
  POrt           : No                No | 1-65535
  HOST           : *
  (Mixed Case)   :
  PAth           : %D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D0%BD%D0%B4%D1%80%D0%
  (Mixed Case)   : A1%D0%BE%D0%BB%D0%B6%D0%B5%D0%BD%D0%B8%D1%86%D1%8B%D0%BD*
```

This example Atom entry contains an IRI using the equivalent XML character references for the Unicode characters that are represented in the example URIMAP resource:

```
<entry>
<link rel="self" href="http://example.com:5050/&#x0410;&#x043B;&#x0435;
&#x043A;&#x0441;&#x0430;&#x043D;&#x0434;&#x0440;&#x0421;&#x043E;&#x043B;&#x0436;
&#x0435;&#x043D;&#x0438;&#x0446;&#x044B;&#x043D;/000100"/>
<id>tag:example.com,2009-02-13:file:FILEA:000100</id>
<title>FILEA item 000100</title>
<rights>Copyright (c) 2009, Joe Bloggs</rights>
<published>2008-11-06T12:35:00.000Z</published>
<author>
 <name>Joe Bloggs</name>
 <email>JBloggs@example.com</email>
</author>
<app:edited>2009-03-11T14:42:38+00:00</app:edited>
<updated>2009-03-11T14:42:38+00:00</updated>

<content type="text/xml">
  <DFH0CFIL xmlns="http://www.ibm.com/xmlns/prod/cics/atom/filea">
  <filerec>
  <numb>000100</numb><name>S. D. BORMAN</name><amount>$0100.11</amount>
  </filerec>
  </DFH0CFIL>
</content>

</entry>
```

Selector value for Atom entries

The selector value for an Atom entry is any identifier that CICS or a service routine can use to locate the record in a file, temporary storage queue, or other resource that contains the data for the Atom entry. A suitable selector value is any identifier that is unique and always applies to a given record in the resource that holds the data for the Atom entries, such as an item number or unique key.

When CICS is issuing an Atom document as a response to a Web client, CICS uses the selector values for the individual Atom entries to construct links directly to the Atom entries, and also as part of the generated Atom IDs for the entries. The Web client can use the links to make requests for single Atom entries. The selector value from each link identifies the correct record in the resource that contains the data for the Atom entries. "URLs for Atom feeds from CICS" on page 245 explains how a selector value is appended to a link.

When CICS is delivering Atom entries directly from a file or temporary storage queue, CICS identifies a suitable selector value depending on the type of resource. For a temporary storage queue, the selector value is the number that identifies the record in the temporary storage queue, which CICS assumes is a decimal number. For a file, the selector value is the key for the file, which must be unique. CICS assumes that the format of the selector value for each file type is as follows:

- A decimal number for RRDS and VRRDS files.

- A binary number for ESDS and extended ESDS files.
- A character string for any other type of VSAM file.

If the key for your file is not in the format that CICS assumes, you can specify the correct format in the <cics:selector> element in the Atom configuration file.

When a service routine returns the data for Atom entries, you can choose what the selector value is. The selector value can be anything that your program can use to locate the correct record in your resource to provide the data for the Atom entry. For example, if the resource is a database, you might use the unique identifier that provides a key for the records. If the key is not a character string, you must specify in the <cics:selector> element in the Atom configuration file that you are using a hexadecimal selector value.

When a service routine returns an Atom entry from a feed or collection, you must use the **ATMP_NEXTSEL** parameter in the DFHATOMPARMS container to return a selector value for the next Atom entry that you have available in the feed. If the Web client has requested a number of entries, CICS links to your program again using this selector value, so that your program can identify and return the next Atom entry that is held as a record in your resource. This process continues until CICS has enough entries for the feed, or until your program returns a null value to indicate that no further Atom entries are available from your resource.

When a service routine returns an Atom entry from a collection, you must use the **ATMP_PREVSEL**, **ATMP_FIRSTSEL**, and **ATMP_LASTSEL** parameters in the DFHATOMPARMS container to return selector values for the previous, first, and last Atom entries in the collection. CICS uses these values to construct <atom:link> elements containing links to other partial lists of entries in the collection. You may return these values for an Atom entry from a feed, if you think they would be useful to your Web clients in order to retrieve other windows of Atom entries from the feed, but they are not required for a feed. The processing to produce a link using **ATMP_PREVSEL** increases response times, so only specify this value for a feed if your Web clients are set up to use this form of navigation.

The identity of the first, previous, next, and last Atom entries in your feed or collection depends on the order in which you choose to return the Atom entries. “Sequence for Atom entries” explains how CICS determines the order in which to return Atom entries, and suggests the order in which a service routine can return Atom entries.

Sequence for Atom entries

CICS, or your service routine, must determine the order in which multiple Atom entries are arranged in an Atom feed document.

RFC 5023, *The Atom Publishing Protocol*, which is available from http://www.ietf.org/rfc/rfc5023.txt, states that entries in an Atom collection should be returned to a Web client according to the order in which they were edited, as shown by the <app:edited> element in the entry. The Atom entry that was most recently edited should be returned first, so that it is the first Atom entry to appear in the Atom feed document. The next most recently edited Atom entry should be returned next, and so on, with the entry that was least recently edited being returned last. This function is a SHOULD requirement in RFC 5023 for a full list of Atom entries, where the whole collection is returned in a single feed document, but a MUST requirement for a partial list of Atom entries. RFC 5023 and RFC 4287 do not make any requirement for the ordering of Atom entries in an

Atom feed that is not defined as a collection, so for entries in an Atom feed, servers can choose any order that is consistent and logical.

For reasons of performance, CICS does not automatically return Atom entries in a collection in the order in which they were most recently edited. CICS deviates from this requirement in order to maintain acceptable response times while still providing the useful function of partial lists. For both Atom feeds and collections, when CICS is extracting data directly from a resource to produce Atom documents, CICS returns Atom entries ordered by the time when they were written as records in the resource, as far as CICS can determine. The Atom entry that was written most recently is returned first, the next most recently written Atom entry is returned next, and so on. CICS determines the order of writing as follows:

- For temporary storage queues, the Atom entry that has the highest record number is returned first.
- For ESDS and extended ESDS files, the Atom entry that has the highest RBA (relative byte address) or XRBA (extended relative byte address) is returned first.
- For RRDS and VRRDS files, the Atom entry that has the highest RRN (relative record number) is returned first.
- For KSDS and AIX files, which do not have a concept of the order of writing, the Atom entries are returned in order of their record key, and the Atom entry with the **lowest** record key is returned first.

If you use a service routine to supply the data for your Atom entries, you can choose the order in which you return the Atom entries. If you want to return Atom entries in a collection according to when they were edited, in compliance with RFC 5023, a service routine can do this for Atom entries that are stored in a file. To return Atom entries in order of editing, take the following actions:

1. In your file, include a field in the records that contains a time stamp in ABSTIME format showing when the entry was last edited. You can output this information in your Atom entries as the <app:edited> element.
2. Define the field containing the time stamp as an alternate index for the file.
3. In your service routine, use the alternate index to locate the records containing the data for the Atom entries, and return them with the most recently edited entry first, as indicated by the most recent timestamp.

You can also use this method if you want to return Atom entries in a feed according to when they were updated, rather than when they were first written. If you cannot store a suitable time stamp in the file that holds the data for your Atom entries, or if you find that ordering the entries using that information produces unacceptable response times, return the Atom entries in any order that is consistent and logical, such as the order used when CICS extracts data directly from a resource.

The values that your service routine supplies for the **ATMP_PREVSEL**, **ATMP_NEXTSEL**, **ATMP_FIRSTSEL**, and **ATMP_LASTSEL** parameters in the DFHATOMPARMS container depend on the order that you have chosen for returning your Atom entries. If you are returning the Atom entries according to when they were edited or updated, as indicated by a time stamp, then the values for the parameters are as follows:

- The previous Atom entry is the Atom entry that was edited after the present entry was edited.
- The next Atom entry is the Atom entry that was edited just before the present entry was edited.
- The first Atom entry is the Atom entry that was edited the most recently.
- The last Atom entry is the Atom entry that was edited the least recently.

If you are returning the Atom entries according to when they were first written, then the values for the parameters are as follows:

- The previous Atom entry is the Atom entry that was written after the present entry was written.
- The next Atom entry is the Atom entry that was written just before the present entry was written.
- The first Atom entry is the Atom entry that was written the most recently.
- The last Atom entry is the Atom entry that was written the least recently.

Date and time stamps for Atom entries

The metadata for an Atom entry can include date and time stamps to show when the Atom entry was first published, when it was last updated, and when it was last edited.

The Atom Syndication Format and Atom Publishing Protocol define these date and time stamps as follows:

<atom:published>
: The date and time when the Atom entry was first created or first made available. For example, if your Atom feed uses records in a database to provide the data for the Atom entries, this date and time would be the point when the record containing the data was added to the database.

<atom:updated>
: The date and time when the Atom entry was last changed in a way that you consider to be significant. For example, you might record this date and time stamp if the value of a field in the record in the database was changed.

<app:edited>
: The date and time when the Atom entry was last edited. This date and time stamp applies only to Atom entries that are part of a collection, and in that case it is required (as a SHOULD requirement).

If you are setting up a new resource to contain data for Atom entries, you can include fields in the records in the resource to hold the date and time stamps. A service routine can return this data by overwriting the **ATMP_PUBLISHED**, **ATMP_UPDATED** , and **ATMP_EDITED** parameters in the DFHATOMPARMS container. If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_PUBLISHED_FLD**, **ATMP_UPDATED_FLD**, and **ATMP_EDITED_FLD** parameters have the name and length of the relevant field in the records in your resource.

If the records in your resource do not contain any fields to hold metadata, CICS provides the current date and time as a default timestamp for all of these elements. In this case, a service routine returns spaces for the relevant parameters in the DHFATOMPARMS container. For the <atom:published> element, you can specify an alternative default timestamp in the prototype Atom entry in your Atom configuration file. You cannot specify alternative defaults for the <atom:updated> and <app:edited> elements in the prototype Atom entry in your Atom configuration file.

The date and time stamps that you use for these elements must be in the RFC 3339 format, also known as the XML dateTime datatype. RFC 3339, *Date and Time on the Internet: Timestamps* , is a format specification for date and time stamps in UTC

(Coordinated Universal Time), taken from the ISO 8601 standard. You can read this specification at http://www.ietf.org/rfc/rfc3339.txt.

Use the EXEC CICS ASKTIME command followed by the EXEC CICS FORMATTIME command to produce a date and time stamp in the RFC 3339 format. Alternatively, if your service routine can use the TRANSFORM DATATOXML command, you can convert a CICS ABSTIME value held in your resource record into a date and time stamp in this format.

If you are populating a record in your resource with data for a new Atom entry supplied by a Web client (a POST request), or editing the fields in a record in your resource at the request of a Web client (a PUT request), the Web client might provide date and time stamps in the <atom:updated>, <atom:published>, or <app:edited> elements. In the case of the <atom:updated> and <app:edited> elements, it is advisable to ignore these and generate a new date and time stamp to ensure accuracy and validity. For a PUT request in particular, the date and time stamps might just be the date and time stamps from the existing record in the resource, returned unchanged.

Related reference:

"DFHATOMPARMS container" on page 271
DFHATOMPARMS is a container of DATATYPE(CHAR) that contains parameters that CICS uses to communicate with a service routine that provides data for an Atom feed.

Atom IDs for Atom entries

Each Atom entry has a unique Atom ID that must remain the same for the lifetime of the Atom entry.

The Atom ID for an Atom entry is specified in the <atom:id> element. It must be in the form of a valid Internationalized Resource Identifier (IRI), but it does not need to relate to a real resource location.

Tag URIs

CICS can generate a unique Atom ID for each Atom entry in the tag URI format when it serves the Atom feed, using information that you specify in the <cics:authority> element in the Atom configuration file. The tag URI scheme is described in RFC 4151, *The 'tag' URI Scheme*.

To produce the tag URI for the Atom ID of an Atom entry, CICS uses the following items in order:

1. A scheme of "tag"
2. An authority name and date that you specify in the <cics:authority> element in the Atom configuration file
3. A specific consisting of the resource type and resource name that you specify in the <cics:resource> element in the Atom configuration file, and the selector value for the individual Atom entry

The authority name and date are separated by a comma, and the other elements are separated by a colon. An example of a tag URI produced by CICS is as follows:

```
tag:example.com,2009-01-08:tsqueue:WB20TSQ:23
```

The authority name in the tag URI is a domain name or email address that is registered to you or to your company, and the date is a date on which the

authority name was owned by you or your company. "<cics:authority> element" on page 303 has details of the requirements for the authority name and date.

For an Atom feed where CICS obtains data directly from a file or temporary storage queue, the resource type and resource name are those of the file or temporary storage queue. For an Atom feed where a user-written service routine provides the data, the resource type and resource name are those of the service routine.

The tag URIs that CICS produces as Atom IDs have the following characteristics:

- The Atom ID remains the same for the lifetime of each Atom entry, as long as you do not change the name of the file, temporary storage queue, or service routine, change the relevant information in the Atom configuration file, or move the Atom entry to a different resource.
- The Atom ID remains the same if the same resource is served in the same way from a different CICS region.
- The Atom ID changes if you rename the file, temporary storage queue, or service routine. To be compliant with the Atom format, if you rename a resource, you must not continue to serve it as the same Atom feed (with the same URL), because its Atom IDs are different.
- The Atom ID is unique within a CICS region, but it is not guaranteed to be unique across different CICS regions. In the situation where you want to set up Atom feeds from resources that have the same name and type but are in different CICS regions, you can specify a different authority name or a different date in the <cics:authority> element of the Atom configuration file for each of the feeds. Tag URIs that have different dates are not equivalent to each other, even if all the other information is the same.
- The Atom ID is unique for Atom entries provided by a user-written service routine that deals with a single Atom feed, but it is not unique if the user-written service routine provides more than one feed. If your user-written service routine provides multiple feeds, either choose an alternative format for your Atom IDs, or use a different authority name or date in the <cics:authority> element of the Atom configuration file for each of the feeds.

Alternative formats for Atom IDs

Instead of using the tag URI format that is generated by CICS, you may specify an alternative format for your Atom IDs using the <atom:id> element for the prototype Atom entry in the Atom configuration file. CICS appends the selector value to your alternative format, to produce a unique Atom ID for each Atom entry.

If you use an alternative Atom ID format, make sure that the resulting Atom IDs are unique and meet the requirements of the Atom format specification in RFC 4287.

To ensure correct formatting, CICS ignores any Atom IDs that are supplied by Web clients, and instead uses the format that you specify in the Atom configuration file for the feed.

Storing Atom IDs

Because CICS can produce the same Atom ID for an Atom entry each time it serves the Atom entry, it is not essential to store the Atom ID with the Atom entry. This function enables you to provide Atom entry data from a resource that does

not contain fields to store metadata, provided that you keep the Atom IDs the same and do not change the Atom ID in the configuration file, move the Atom entry to a different resource, or, for tag URIs, change the name of the resource or service routine.

However, RFC 4287 recommends that an Atom ID should be stored with the Atom entry. If you are able to store Atom IDs in the resource that holds the data for your Atom entries, you can follow this recommendation. If you are storing your Atom entries in a file, this field can be the unique key for the records. CICS, or your service routine, stores a complete Atom ID for the Atom entry in the field, and an Atom ID stored with an Atom entry can differ from and override the Atom ID that CICS would generate for that Atom entry.

For a service routine, CICS uses the **ATMP_ATOMID** parameter to send a prototype Atom ID for the Atom entry, using the information that you specified in either the <cics:authority> element or the <atom:id> element in the Atom configuration file. To produce a complete Atom ID, your service routine can either complete the prototype Atom ID by appending the selector value, or ignore it and substitute its own valid Atom ID. For example, you could generate a URI with the urn:uuid scheme using a hexadecimal Universally Unique Identifier (UUID), as described in RFC 4122, *A Universally Unique IDentifier (UUID) URN Namespace*. The service routine can store the Atom ID in the resource record, using the field named in the **ATMP_ID_FLD** parameter, and then return it using the **ATMP_ATOMID** parameter.

To ensure accuracy, CICS ignores Atom IDs that are supplied by Web clients, and does not store these in the records in a file or temporary storage queue, or pass them to a service routine.

RFC 4287 requires that the Atom ID remain with the Atom entry if the entry is reused or moved to another location. If you store Atom IDs with your Atom entries, you can move the Atom entries to another location and still comply with this requirement. If you do not store Atom IDs with your Atom entries, do not move the Atom entry to another location.

Atom IDs for Atom feeds

An Atom feed also has a unique identifier. If you use the <cics:authority> element in the Atom configuration file to make CICS generate tag URIs as Atom IDs, CICS generates an Atom ID for the Atom feed in the same format as for the Atom entries, but without the selector value or unique identifier that is appended for the Atom entries. For example:

```
tag:example.com,2009-01-08:tsqueue:WB20TSQ
```

If you prefer an alternative Atom ID format, you can use the <atom:id> element for the Atom feed to specify a complete Atom ID for the Atom feed. Make sure that the Atom ID is unique and meets the requirements of the Atom format specification in RFC 4287.

Related reference:

"<cics:authority> element" on page 303
The <cics:authority> element in an Atom configuration file provides the authority name and associated date that CICS uses when creating tag URIs to use as Atom IDs for individual Atom entries.

Chapter 21. CICS samples for Atom feeds

CICS supplies sample URIMAP and ATOMSERVICE resource definitions, Atom configuration files, XML bindings and service routines.

The sample service routines for Atom feeds are in the SDFHSAMP sample library. The sample resources for Atom feeds are in two locations:

- The CICS resource group DFH$WEB2.
- The `/samples/web2.0/` subdirectory of the root directory for CICS files on z/OS UNIX, as specified by the CICS system initialization parameter USSHOME. The default value for USSHOME is `/usr/lpp/cicsts/cicsts41`.

The CICS resources in the resource group DFH$WEB2 reference the files in the `/samples/web2.0/` subdirectory using the path `/usr/lpp/cicsts/cicsts41/samples/web2.0/`, with the default value for USSHOME. If the USSHOME system initialization parameter for your CICS region specifies a nondefault root directory for CICS files on z/OS UNIX, to use the samples you must complete these steps:

1. Copy the resource group DFH$WEB2 to a new resource group.
2. Modify the URIMAP resource definitions in your copy of the DFH$WEB2 group to change the occurrences of the default directory `/usr/lpp/cicsts/cicsts41` in the HFSFILE attribute to the name of the root directory that your CICS region uses for CICS files on z/OS UNIX.
3. Modify the ATOMSERVICE resource definitions in your copy of the DFH$WEB2 group to change the occurrences of the default directory `/usr/lpp/cicsts/cicsts41` in the CONFIGFILE and BINDFILE attributes to the name of the root directory that your CICS region uses for CICS files on z/OS UNIX.

Sample Atom collection

The sample Atom collection demonstrates how to create an Atom collection to contain data about employees, including their geographic location. Instructions to set up and use the sample Atom collection are in the Web 2.0 scenario "Create an Atom feed to work with employee information" in the CICS TS for z/OS, Version 4.1 Information Center, which is available at `https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp`.

The components shown in the following table are used to produce the sample Atom collection.

Table 8. Components for the sample Atom collection

Component	Purpose	Location
URIMAP resource DFH$W2Q1	Handles HTTP requests for the sample Atom collection and for individual Atom entries	CICS group DFH$WEB2 or your copy
ATOMSERVICE resource DFH$W2Q1	Names resources used to produce the Atom collection	CICS group DFH$WEB2 or your copy
Atom configuration file `dfh0w2q1.xml`	Provides metadata and structure for the Atom feed document and Atom entries	*USSHOME*`/samples/web2.0/atom`

Table 8. Components for the sample Atom collection (continued)

Component	Purpose	Location
XML binding `dfh0w2q1.xsdbind` and XML schema `dfh0w2q1.xsd`	Inform CICS about the structure of the records in the temporary storage queue	*USSHOME*`/samples/web2.0/atom`
URIMAP resource DFH$W2AC	Handles HTTP requests to view the Atom configuration file or XML binding in a Web browser	CICS group DFH$WEB2 or your copy
COBOL language structure (copybook) DFH0W2Q1	Describes the structure of the records in the temporary storage queue; was used to produce the XML binding	SDFHSAMP sample library
CICS icon `cics-icon.gif` and CICS logo `cics-logo.gif`	Icon and logo images for the sample Atom collection	*USSHOME*`/samples/web2.0/image`
URIMAP resource DFH$W2GI	Handles HTTP requests for the icon and logo images	CICS group DFH$WEB2 or your copy

The components shown in the following table are used to produce the mashup Web page that enables you to interact with the sample Atom collection.

Table 9. Components for the mashup Web page

Component	Purpose	Location
Mashup Web page `dfh$w2q1.html`	A mashup that lets you interact with the sample Atom collection	*USSHOME*`/samples/web2.0/html`
URIMAP resource DFH$W2HT	Handles HTTP requests for the mashup Web page	CICS group DFH$WEB2 or your copy
Stylesheet `dfh$w2ss.css`	Controls the appearance of the mashup Web page	*USSHOME*`/samples/web2.0/style`
URIMAP resource DFH$W2SS	Handles HTTP requests for the stylesheet	CICS group DFH$WEB2 or your copy
Script `dfh$w2w2.js`	A widget, used in the mashup Web page, that displays and receives the data from the Atom entries	*USSHOME*`/samples/web2.0/script`
URIMAP resource DFH$W2JS	Handles HTTP requests for the script dfh$w2w2.js	CICS group DFH$WEB2 or your copy

When you set up the sample Atom collection using these components, the URL for Web client requests for the collection is as follows:

`http://`*`host`*`:`*`port`*`/atom/q/personnel/feed`

where:

- *`host`* is the character host name, IPv4 address, or IPv6 address that is defined in the HOST attribute of the TCPIPSERVICE resource that you are using with this collection.
- *`port`* is the port number that is defined in the PORTNUMBER attribute of the TCPIPSERVICE resource that you are using with this collection.

The URL for the mashup Web page that displays the sample Atom collection is as follows:

```
http://host:port/web2.0/html/dfh$w2q1.html
```

where *host* and *port* are the host name and port number from the TCPIPSERVICE resource that you are using with this collection.

DFH0W2F1 COBOL sample service routine

The sample service routine DFH0W2F1 is a COBOL program that demonstrates how a user-written service routine can handle GET, POST, PUT, and DELETE requests for Atom entries that use data from the CICS sample file FILEA. For information about this service routine and instructions to run it, see "DFH0W2F1 COBOL sample service routine for Atom feeds" on page 353.

CICS supplies the components shown in the following table to run DFH0W2F1.

Table 10. Components to run DFH0W2F1

Component	Purpose	Location
URIMAP resource DFH$W2P1	Handles HTTP requests for an Atom feed using FILEA, with DFH0W2F1 as the service routine	CICS group DFH$WEB2 or your copy
ATOMSERVICE resource DFH$W2P1	Names resources used to produce the Atom feed	CICS group DFH$WEB2 or your copy
Atom configuration file `dfh0w2f1.xml`	Provides metadata and structure for the Atom feed document and Atom entries	*USSHOME*`/samples/web2.0/atom`

The ATOMSERVICE resource DFH$W2P1 does not name an XML binding for the FILEA file, because FILEA does not contain any fields that can be used as metadata for Atom entries, so DFH0W2F1 does not make use of the <cics:fieldnames> element in the Atom configuration file.

When you set up an Atom feed using these components, the URL for Web client requests for the Atom feed is as follows:

```
http://host:port/atom/p/filea/feed
```

where *host* and *port* are the host name and port number from the TCPIPSERVICE resource that you are using with this Atom feed.

DFH$W2S1 C sample service routine

The sample service routine DFH$W2S1 is a C language program that demonstrates how a user-written service routine can read the parameters in the DFHATOMPARMS container, update the metadata and content containers (such as DFHATOMTITLE and DFHATOMCONTENT), and update and return the DFHATOMPARMS container.

As shipped, DFH$W2S1 can respond to GET requests for an Atom feed by returning the default data that is set up by its code. It does not demonstrate the process of identifying the required record from the resource that holds the data for your Atom entries, and extracting the appropriate fields from the record, or

updating the record in response to a POST, PUT, or DELETE request. You can use DFH$W2S1 to supply default data for test purposes in response to GET requests for an Atom feed.

CICS does not supply resources to run DFH$W2S1. To run DFH$W2S1, you must create and install an appropriate RDO definition, which must specify EXECKEY(CICS), for DFH$W2S1, and set up URIMAP and ATOMSERVICE resources and an Atom configuration file that specify DFH$W2S1 as the service routine.

For details of how DFH$W2S1 works, see "DFH$W2S1 C sample service routine for Atom feeds" on page 287.

Sample Atom feed from FILEA

CICS provides a set of resources to serve the CICS sample file FILEA as an Atom feed directly from CICS.

The components shown in the following table are used to serve this Atom feed.

Table 11. Components to run DFH0W2F1

Component	Purpose	Location
URIMAP resource DFH$W2F1	Handles HTTP requests for an Atom feed using FILEA	CICS group DFH$WEB2 or your copy
ATOMSERVICE resource DFH$W2F1	Names resources used to produce the Atom feed	CICS group DFH$WEB2 or your copy
Atom configuration file `filea.xml`	Provides metadata and structure for the Atom feed document and Atom entries	*USSHOME*`/samples/web2.0/atom`
XML binding `filea.xsdbind` and XML schema `filea.xsd`	Inform CICS about the structure of the records in FILEA	*USSHOME*`/samples/web2.0/atom`

When you set up an Atom feed using these components, the URL for Web client requests for the Atom feed is as follows:

`http://`*host*`:`*port*`/atom/f/filea/feed`

where *host* and *port* are the host name and port number from the TCPIPSERVICE resource that you are using with this Atom feed.

Chapter 22. Setting up a resource to supply Atom entry data

An Atom feed or collection consists of a series of Atom entries, which are items of data together with suitable metadata. For an Atom feed served by CICS, the data for the Atom entries is taken from the records in a resource, which could be a file, a temporary storage queue, or another resource such as a database table. A single record provides a single Atom entry.

About this task

A record in your resource might hold items of metadata for the Atom entry as well as the content for the Atom entry, or it might hold only the content for the Atom entry. When you set up your Atom feed, you can make CICS supply any required items of metadata that are not held in your resource records.

You can use any of these resources to supply the data for the Atom entries in your Atom feed:

- A new VSAM file or temporary storage queue that you create to contain Atom entries.
- An existing VSAM file or temporary storage queue that is defined to CICS, from which CICS can extract data directly to produce the Atom feed. CICS can extract data for Atom feeds from any type of VSAM file, except for an alternate index file that has been defined with the NONUNIQUEKEY attribute. The file must have a unique key for its records. CICS cannot extract data for Atom feeds directly from BDAM files.
- Any other resource that you can access from a CICS application program. You can deliver a CICS or non-CICS resource using a CICS application program known as a service routine, which extracts data for Atom entries from the resource and supplies it to CICS in containers.

For ESDS files,

Procedure

- To create a new VSAM file or temporary storage queue to contain Atom entries, follow the instructions in “Creating a CICS resource to store Atom entries” on page 264.
- To expose an existing VSAM file or temporary storage queue as an Atom feed:
  1. Find, or write, a language structure that describes the structure of the records in the resource.
     - You can use a high-level language structure, or copybook, in COBOL, C, C++, or PL/I. The language structure must be in a partitioned data set. For a file or temporary storage queue that is used by a CICS application program, a language structure should already exist. You can write a language structure for the records if you do not already have one.
     - Alternatively, you can use an XML schema or WSDL document that describes the structure of the records in the resource.
  2. Use your language structure to produce an XML binding for the resource, as described in "Generate mappings from language structures" in the *CICS Application Programming Guide*. If you have an XML schema or WSDL document instead, follow the alternative instructions in "Generate mappings from an XML schema" in the *CICS Application Programming Guide*.

- To deliver any other CICS or non-CICS resource, write a service routine to extract data for each Atom entry from a record in the resource, and supply the data to CICS in a set of containers. For instructions to write a service routine, see "Writing a program to supply Atom entry data" on page 268.

What to do next

When you have chosen the resource that holds your Atom entry data, and created an XML binding or a service routine to support the delivery of this data, set up your Atom feed following the instructions in Chapter 23, "Setting up CICS definitions for an Atom feed," on page 293.

ESDS files with Atom feeds

You may use ESDS (entry-sequenced data set) files to hold Atom entry data for an Atom feed, but there are some restrictions on deleting the Atom entries, which apply if you set up your feed as an editable collection.

Web clients can delete Atom entries in a collection by making HTTP requests with the DELETE method. With an ESDS file, HTTP requests with the DELETE method are only supported if the ESDS has no alternate index defined.

In response to a DELETE request, CICS deletes the relevant record from the ESDS by rewriting it with 'FF'x as the first byte, to represent a logical deletion. If Web clients make subsequent HTTP requests with the GET method to retrieve the Atom entry that was in the deleted record, CICS returns a "not found" response to the GET requests.

When you define an ESDS file as an Atom collection, you must use one of the following methods to ensure that other application programs that use the ESDS file handle the deleted records correctly:

- In the FILE resource definition for the ESDS, set DELETE to NO.
- Alternatively, code the applications to process a record beginning with 'FF'x as being logically deleted.

To avoid these restrictions, if you are setting up a new resource to store Atom entry data for a collection, choose a VSAM file type other than ESDS.

If the ESDS file is only used for an Atom feed that is not defined as a collection, so Web clients cannot make requests with the DELETE method, these restrictions do not apply. However, if you are setting up a new resource to store Atom entry data for an Atom feed, avoid using an ESDS file in case you decide to set up the Atom feed as a collection later on.

Creating a CICS resource to store Atom entries

To store data as Atom entries, create a file or temporary storage queue in CICS, and write a language structure in COBOL, C, C++, or PL/I to explain its structure.

About this task

In your new file or temporary storage queue, each record represents a single Atom entry. Each field in the record contains the data for a single element in the Atom entry, which can be its content or an item of metadata such as its title. When you set up your Atom feed, you specify the names of these fields to CICS using the

<cics:fieldnames> element in the Atom configuration file, and CICS will extract the data from each record to assemble an Atom entry.

The complete listing and description of the possible elements in an Atom entry is in RFC 4287, *The Atom Syndication Format*, which is available from http://www.ietf.org/rfc/rfc4287.txt. CICS does not support all of these elements, and, of the elements that CICS does support, some are not supported in your file or temporary storage queue, but can only be specified in the Atom configuration file. For a list and description of the elements that CICS supports in files and temporary storage queues, see "<cics:fieldnames> element" on page 306. For a complete list of the elements that CICS does and does not support in Atom feeds and Atom entries, see "Atom element reference for CICS" on page 316.

Procedure

1. Decide whether to use a temporary storage queue or a file as the resource to store the data for your Atom entries.
   - A temporary storage queue is suitable if you are experimenting with Atom feeds in CICS, because you do not have to define a temporary storage queue to CICS before you use it, although you will have to set up a CICS resource definition if you want to apply security measures. It is also suitable for an Atom feed where the Atom entries are not of long-term interest; for example, if you are issuing alerts for events in an application.
   - A file takes longer to set up than a temporary storage queue, because you must define a file to CICS before you can use it, and it normally requires a physical data set. However, a file provides suitable long-term storage for any Atom feed, including a feed that you might want to set up as an editable collection. A file that holds Atom entries must have a unique key for the records, and you cannot use an alternate index file that has been defined with the NONUNIQUEKEY attribute. You can use any type of VSAM file to hold Atom entries, but note that ESDS (entry-sequenced data set) files are not a good choice for a feed that you might want to set up as an editable collection, for the reasons mentioned in "ESDS files with Atom feeds" on page 264. You cannot use a BDAM file.
2. Plan the content of the records in your file or temporary storage queue. The content of the Atom entry is the only item that CICS requires in your records, because you can specify all the metadata in the Atom configuration file. However, when you are setting up a dedicated file or temporary storage queue to contain Atom entries, you can include fields for metadata in the records, which you can use to provide metadata specific to each Atom entry. The following list summarizes the items of data that you can include as fields in your records and whether they are required or optional:

   Atom ID

   A unique identifier for the Atom entry. For more information about the format of Atom IDs, see "Atom IDs for Atom entries" on page 255.

   Atom entries must have a unique Atom ID. CICS can generate a unique Atom ID for each entry when it serves the Atom feed, using information that you specify in the Atom configuration file. An Atom ID created by CICS remains the same for the lifetime of the Atom entry as long as you do not change the name of the file or temporary storage queue, change the relevant information in the Atom configuration file, or move the Atom entry to a different resource. You therefore do not have to include a field in your records to store the Atom ID.

However, to comply fully with the Atom format, an Atom ID must remain with the entry if the entry is reused or moved to another location. If you think that you might use your Atom entries anywhere other than in this file or temporary storage queue, or if you just prefer to follow the recommendation in RFC 4287 that an Atom ID should be stored with the entry, include a field in your records to hold the Atom ID. If you are storing your Atom entries in a file, this field can be the unique key for the records.

Author's details
The personal name, email address, and Web site of the principal author of the Atom entry, in three separate fields. You must supply an author name either for the Atom feed or for all the Atom entries, but the other fields are optional. If your Atom entries have different authors, include a field in your records for the name, and fields for other details if you want. If the name and other details of the author are the same for all your Atom entries, specify the name and details in the Atom configuration file instead.

Category
A category term that classifies the entry. This field is optional. If you plan to set up this Atom feed as an editable collection, and to use categories to describe your collection, include this field. If you do not plan to set up this feed as a collection, you can still include the field if it might be helpful to consumers of your feed.

Content
The entire content to be published in the Atom entry. CICS requires content for every Atom entry. Your content can be plain text, or HTML, XHTML, XML, or another text media type. CICS does not support nontext content, or Atom entries with no content. If you are including any fields for metadata, you must have a field, or a substructure of nested fields in the record, that holds the content. If you are not including any fields for metadata, CICS publishes the whole of the record from the file or temporary storage queue as the content of the entry.

Content type
The media type for the content of the Atom entry, such as text or XML. This field is optional. If all your Atom entries have the same type of content, you can specify the media type in the Atom configuration file instead.

Date last edited
The time stamp that indicates when the record was last edited. You can use time stamps in the XML dateTime format, as described in RFC 3339, or a CICS ABSTIME value. For more information about date and time stamps, see "Date and time stamps for Atom entries" on page 254. If you plan to set up this Atom feed as an editable collection, including this field enables you to return the Atom entries according to when they were last edited, which is recommended by the Atom Publishing Protocol for a collection. If you do not plan to set up this feed as a collection, do not include this field.

Date first published
The time stamp or ABSTIME value that indicates when the record was first created or published as an Atom entry. This field is optional. If you think that it might be helpful to consumers of your feed, include the field.

Title The title for the Atom entry. CICS only supports plain text for titles. A title is required for each Atom entry, so you normally need to include this field. If all your Atom entries have the same title, you can specify this title in the Atom configuration file instead.

Summary
A summary of the Atom entry. CICS only supports plain text for summaries. This field is optional unless the content of the entry is a nontext media type, in which case a summary is required. CICS does not provide any support for nontext content in Atom entries.

Date last updated
The time stamp or ABSTIME value that indicates when the record was last updated. An updated time stamp is required for each Atom entry, so you normally need to include this field. If you cannot include this time stamp or ABSTIME value in your file or temporary storage queue, you can omit the field and CICS can supply the current date and time when it issues the entry in an Atom feed document, or a suitable alternative default that you specify in the Atom configuration file.

3. Write a language structure, or copybook, in COBOL, C, C++, or PL/I for your file or temporary storage queue. A language structure describes the fields in a record in your file or temporary storage queue, stating the name, content type, and length of each field in the order in which they appear. If you plan to create records in your file or temporary storage queue using an application in COBOL, C, C++, or PL/I, the application uses this language structure to write to the file or temporary storage queue. You also need this language structure to produce an XML binding for the file or temporary storage queue, so it is required even if your application is in a different language, or if you are not using an application, for example, if you are experimenting with Atom feeds in CICS and your record structure is simple enough that you can use the CECI transaction to write to the file or temporary storage queue.

   Note: Make sure that fields that are used to provide metadata for Atom entries are not nested in your language structure. The metadata fields in your record must all be listed in your language structure at the same level. You may use structures of nested fields within the field that provides the content for the Atom entry.
   Store your language structure in a partitioned data set that has a fixed record length of 80 bytes. This example COBOL language structure declares alphanumeric fields of appropriate lengths to contain the data for each element:

```
******************************************************************
* Name: SAMPBIND.cob                                             *
*                                                                *
*                                                                *
* This is a COBOL copy book to describe the data record.         *
* You can generate a binding file from this.                     *
*                                                                *
******************************************************************

     03 TITLE-FIELD                    PIC X(50).
     03 SUMMARY-FIELD                  PIC X(500).
     03 ATOMID-FIELD                   PIC X(20).
     03 CONTENT-FIELD                  PIC X(500).
     03 AUTHOR-NAME-FIELD              PIC X(30).
     03 AUTHOR-EMAIL-FIELD             PIC X(256).
     03 AUTHOR-URI-FIELD               PIC X(256).
     03 EDITED-FIELD                   PIC X(25).
     03 UPDATED-FIELD                  PIC X(25).
     03 PUBLISHED-FIELD                PIC X(25).
     03 CATEGORY-FIELD                 PIC X(20).
```

4. Use your language structure as input to the CICS XML assistant to create an XML binding, following the steps in the *CICS Application Programming Guide*.
5. If you have decided to use a file to store your Atom entries:
   a. Set up a suitable VSAM data set, following the procedures in the *CICS System Definition Guide*.
   b. Define the file to CICS by creating and installing a FILE resource definition, using the information in the *CICS Resource Definition Guide*.
6. If you have decided to use a temporary storage queue to store your Atom entries, and you want to specify security and recovery settings for it, define a temporary storage model (TSMODEL) using the information in the *CICS Resource Definition Guide*.

What to do next

If you already have an application that can work with the records in your file or temporary storage queue, test your setup by using your application, or another suitable method, to write at least one record to your file or temporary storage queue, using the WRITEQ TS command for a temporary storage queue, or the WRITE command for a file.

Writing a program to supply Atom entry data

You can write a service routine to provide an Atom feed from any data that can be accessed by a CICS program, such as records from a DB2 database, records in a file, or a COMMAREA. These instructions tell you how to write a program that responds to HTTP GET requests for an Atom feed.

About this task

Web clients might request a number of Atom entries from a feed, or request a specific entry. CICS receives the requests from Web clients, and links to the program with information about each client request. CICS links to the program once for each Atom entry that the client requests, and the program returns a single entry each time.

The program supplies the Atom entry using data that it has extracted from a record in the resource, such as a database or file, that holds the data for the Atom entries for this feed. For an overview of this process, see "Data processing for Atom feeds from CICS" on page 243.

CICS uses a container interface to communicate with the service routine. Use the EXEC CICS GET CONTAINER and EXEC CICS PUT CONTAINER commands to interact with the containers. The C language sample service routine DFH$W2S1 shows you how to use the containers to respond to HTTP GET requests. The COBOL sample service routine DFH0W2F1 also shows you how to use the containers, but be aware that the DFH0W2F1 sample is more complex because it responds to HTTP PUT, POST, and DELETE requests as well as GET requests.

Because the Web client request is an HTTP request, you can also interact with it using the CICS Web API commands, such as the WEB READ HTTPHEADER and WEB READ QUERYPARM commands. If you know how to use these commands, you may use them in the service routine to obtain information directly from the Web client request, including any information that CICS does not provide in the DFHATOMPARMS container.

If you can create an XML binding for the resource that contains the data for your Atom entries, you can pass information from CICS to the program in the DFHATOMPARMS container about the name and length of fields in the resource records that contain data for the Atom entries. Your program can use this information to locate the metadata fields in the resource records. By using these resource handling parameters, you can create a generic service routine that can handle multiple resources. However, you do not have to use the resource handling parameters; if you prefer, you can code information about the resource structure directly in the program.

To respond to a GET request, your service routine must perform these tasks:

Procedure

1. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHATOMPARMS container. CICS uses this container to provide the service routine with information about the request. The sample service routine DFH$W2S1 shows you how to read the parameters in DFHATOMPARMS. “DFHATOMPARMS container” on page 271 has the full documentation for the parameters that CICS passes in this container.
2. Check the value of the **ATMP_HTTPMETH** parameter to verify that the request method is GET. CICS returns an error or makes an appropriate response for methods other than GET, POST, PUT, and DELETE. Instructions for responding to HTTP PUT, POST, or DELETE requests to an Atom collection are not included here, but are given in “How to handle Atom collection editing requests in your service routine” on page 346
3. Use the values of the **ATMP_ATOMTYPE** and **ATMP_SELECTOR** parameters from the DFHATOMPARMS container to identify the record in the resource that contains the data for the Atom entry that the program must return to CICS. The **ATMP_SELECTOR** parameter might contain a selector value that identifies a particular Atom entry. “Selector value for Atom entries” on page 251 explains what the selector value can be, and how CICS and the service routine use it.
   a. If **ATMP_SELECTOR** is null and **ATMP_ATOMTYPE** has the value "feed", the client did not specify a particular Atom entry, so locate the record in the resource that holds the most recent Atom entry that was added to the feed. For example, if the data for your Atom entries is held in a database, use the newest record that was added to the database.
   b. If **ATMP_SELECTOR** contains a selector value and **ATMP_ATOMTYPE** has the value "feed", locate the record in the resource that is identified by the selector value. This combination of values might indicate that CICS needs a second or subsequent Atom entry from the feed to complete a client request, and CICS is requesting one of these Atom entries using a selector value that the service routine supplied in a previous iteration. This combination of values is also used for the first Atom entry in a request when the client has requested a feed document containing a specific range of Atom entries, such as a partial list.
   c. If **ATMP_SELECTOR** contains a selector value and **ATMP_ATOMTYPE** has the value "entry", locate the record in the resource that is identified by the selector value. This combination of values indicates that the client is requesting a single, known Atom entry from the feed.
4. If you have an XML binding for the resource that contains the data for your Atom entries, and you want to use the resource handling parameters to pass information about the fields in the resource, code the service routine to use the values of the **ATMP_TITLE_FLD** parameter and the other parameters ending in **_FLD** to identify the name and length of each field that contains data for an

element of an Atom entry. When you set up an Atom configuration file for the Atom feed that uses data from the resource, you will need to specify the names of these fields in the <cics:fieldnames> element of the Atom configuration file, and CICS will pass them to the service routine using the resource handling parameters. "DFHATOMPARMS container" on page 271 documents the resource handling parameters.

5. Use the PUT CONTAINER command to create a container named DFHATOMCONTENT, with DATATYPE(CHAR), that contains the content for the Atom entry, as stated in the record that you have identified from the resource. This container is required. The sample service routine DFH$W2S1 shows you how to update the container, and "DFHATOMCONTENT container" on page 282 explains what to put in the container.
6. If the record that you have identified from the resource includes any fields that supply metadata for the Atom entry, such as a title, use optional containers to return this metadata to CICS, following the steps in "Returning Atom entry metadata in containers" on page 284.
7. If the record that you have identified from the resource includes any fields that supply date and time stamps for the point when the data was created or updated, return them as new values for the **ATMP_PUBLISHED** and **ATMP_UPDATED** parameters in the DFHATOMPARMS container. The sample service routine DFH$W2S1 shows you how to return new values for these parameters. For information about the format of these date and time stamps, see "Date and time stamps for Atom entries" on page 254.
8. If the **ATMP_SELECTOR** parameter in the DFHATOMPARMS container was null on input to the service routine, meaning that the Web client did not request a specific Atom entry, replace the null value with a suitable selector value for the present entry that you are returning. The sample service routine DFH$W2S1 shows you how to return a selector value if the **ATMP_SELECTOR** parameter is null. "Selector value for Atom entries" on page 251 explains how to choose a selector value. If the **ATMP_SELECTOR** parameter contained a selector value on input to the service routine, do not change it.
9. If the **ATMP_ATOMTYPE** parameter in the DFHATOMPARMS container had the value "feed", indicating that the client wants multiple entries, check whether the resource contains any more, older, data that can be used to provide further Atom entries.
   a. If older data is present, locate the next data item that provides an Atom entry and return a suitable selector value for this data item to be used in the **ATMP_NEXTSEL** parameter. "Sequence for Atom entries" on page 252 explains the order in which you should return the Atom entries.
   b. If no more data is available, set the current length of the data in the **ATMP_NEXTSEL** parameter to zero to return a null value.
10. Read the **ATMP_ATOMID** parameter in the DFHATOMPARMS container to see the prototype Atom ID for the entry. The prototype Atom ID must be completed by appending the selector value for the Atom entry, as specified in the **ATMP_SELECTOR** parameter. If you prefer, your service routine can ignore the prototype Atom ID and substitute its own valid Atom ID for the Atom entry. For more information about the requirements for Atom IDs, see "Atom IDs for Atom entries" on page 255.
    a. If you have stored a complete Atom ID in the resource record for this Atom entry, return this Atom ID followed by its length in the **ATMP_ATOMID** parameter. If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_ID_FLD** parameter has the name and length of the relevant field in the resource.

b. If the resource does not store Atom IDs, set the current length of the data for the **ATMP_ATOMID** parameter to zero. CICS appends the selector value to produce the complete Atom ID.

11. Return a suitable response code to be used in the **ATMP_RESPONSE** parameter in the DFHATOMPARMS container. The sample service routine DFH$W2S1 shows you how to do this. The code is initialized to zero, indicating successful completion. If an error response is returned, CICS produces a suitable default HTTP error response to send to the Web client. "ATMP_RESPONSE parameter in DFHATOMPARMS container" on page 281 lists the available response codes and the HTTP error response that CICS sends in each case. The sample service routine DFH$W2S1 returns control to CICS after setting the response code.

What to do next

When you have written your service routine, create and install a suitable PROGRAM resource definition in CICS to describe the service routine. In your PROGRAM resource definition, use the EXECKEY(USER) attribute. You will need to name this PROGRAM resource in the ATOMSERVICE resource definition for your Atom feed.

When you have set up CICS definitions that use your service routine to provide data for an Atom feed, you can use the CEDX transaction to monitor and debug your service routine as it responds to HTTP requests. CW2A is the default alias transaction for Atom feeds, and your service routine runs under this transaction unless you set up an alternative alias transaction. CEDX monitors the next instance of the transaction that you specify, so if other users are working with Atom feeds in this CICS region using the same alias transaction, set up an alternative alias transaction to use while you are debugging your service routine.

DFHATOMPARMS container

DFHATOMPARMS is a container of DATATYPE(CHAR) that contains parameters that CICS uses to communicate with a service routine that provides data for an Atom feed.

The DFHW2AP series of copybooks map the parameters passed in the DFHATOMPARMS container to the service routine. The following copybooks are defined:

- DFHW2APD for Assembler
- DFHW2APH for C
- DFHW2APL for PL/I
- DFHW2APO for Cobol

The DFHW2CN series of copybooks contain constant values that are referenced by the DFHW2AP series of copybooks. The following copybooks are defined:

- DFHW2CND for Assembler
- DFHW2CNO for Cobol
- DFHW2CNH for C
- DFHW2CNL for PL/I

Input-only parameters in DFHATOMPARMS container

CICS uses these parameters to supply information to the service routine about the Web client's request. These parameters include the resource handling parameters such as **ATMP_TITLE_FLD**.

Each of the input-only parameters in the DFHATOMPARMS container is the address of a double word containing a pointer to an area and the current length of the data in the area. Your service routine must not change these pointers, lengths, or storage.

The parameters ending in **FLD** are used for handling resources. CICS uses these resource handling parameters to supply information about the fields in resource records to a CICS-supplied service routine that is handling a CICS resource such as a temporary storage queue. CICS obtains the names of the fields from the attributes that you specify for the <atom:content> element and the <cics:fieldnames> element in the Atom configuration file, and from the XML binding for the resource. You can use these parameters if you want to write a service routine that obtains its information about resource structures from the Atom configuration file, rather than having this information coded directly in the service routine. If you use these parameters, you must create an XML binding for the resource that contains the data.

ATMP_RESNAME
The name of the CICS resource that supplies the data for the Atom feed. For your service routine, this is always the name of the service routine. CICS requires this parameter for the CICS-supplied service routines that handle various resources directly. CICS obtains this information from the cics:resource attribute of the <atom:content> element.

ATMP_RESTYPE
The type of the CICS resource in uppercase. The resource type can be PROGRAM, TSQUEUE, or FILE. For your service routine, the resource type is always PROGRAM. CICS requires this parameter for the CICS-supplied service routines that handle various resources directly. CICS obtains this information from the cics:type attribute of the <atom:content> element.

ATMP_ATOMTYPE
The type of Atom document being processed, in lowercase. The value of the type string is “feed”, “collection”, or “entry”. “feed” indicates that the client has requested a number of entries from an Atom feed. “collection” indicates that the client has requested a listing of entries in a collection. “entry” indicates that the client has requested a single, specified Atom entry, in either a feed or a collection.

ATMP_HTTPMETH
The HTTP method for the client request, padded. The HTTP method is one of GET, POST, PUT, or DELETE.

ATMP_TAG_AUTHORITY
The authority name specified in the name attribute of the <cics:authority> element in the Atom configuration file. The authority name is a fully qualified domain name or email address that can be used to construct tag URIs. The authority name forms part of the prototype Atom ID if you have selected this format.

ATMP_TAG_DATE
The date specified in the date attribute of the <cics:authority> element in the

Atom configuration file. The date is used with the authority name to construct tag URIs. The date forms part of the prototype Atom ID if you have selected this format.

ATMP_XMLTRANSFORM
The name of an XMLTRANSFORM resource. The XMLTRANSFORM resource is created when you produce the XML binding for a CICS resource and install an ATOMSERVICE resource definition that specifies it. The XMLTRANSFORM resource describes the layout of the records in the resource as an XML structure. If the length of this name is zero, no XML binding was created for the resource, and the service routine must perform its own mapping between the resource records and the elements of the Atom entries.

ATMP_ROOT_ELEMENT
The name of the root element of the XML structure that is mapped by the XMLTRANSFORM resource.

ATMP_MTYPEIN
The media type of the body of the Web client's HTTP request. A request body is present only when the HTTP method, as specified by the **ATMP_HTTPMETH** parameter, is POST or PUT. The media type is always "application/atom+xml", which indicates an Atom entry. CICS passes the request body to the service routine in the DFHREQUEST container. GET and DELETE requests have no request body, so for these HTTP methods the pointer and length are both zero.

ATMP_MTYPEOUT
The media type for the expected content of the Atom entry, as specified in the type attribute of the <atom:content> element in the Atom configuration file for the Atom feed. As in RFC 4287, the media type "text" is used for plain text instead of the IANA media type "text/plain", "html" is used instead of "text/html", and "xhtml" is used instead of "application/xhtml+xml". If the Atom configuration file does not contain this information, CICS passes the default media type "application/xml" to the service routine. The service routine can use the media type to determine suitable markup for the data that it returns in the DFHATOMCONTENT container. If you are using the resource handling parameters and you have a field in your resource records to store a media type for individual Atom entries, the **ATMP_CONTENT_TYPE_FLD** parameter contains the name of this field.

ATMP_WINSIZE
The feed window size. The value is a numeric string that contains either the default number of entries to be returned in each feed or an alternative number of entries that the Web client has requested. This parameter is for information only, because CICS makes a series of requests to the service routine for individual entries.

ATMP_ID_FLD
The name of a field in your resource records that contains the Atom ID of the Atom entry. CICS obtains the name of the field from the atomid attribute of the <cics:fieldnames> element in the Atom configuration file for the Atom feed. If CICS passes this information to the service routine, the service routine can use this named field to store or locate the Atom ID for the entry, and return it in the **ATMP_ATOMID** parameter. This data is used in the <atom:id> element for the entry.

ATMP_PUBLISHED_FLD
The name of a field in your resource records that contains the time when the resource was last published. CICS obtains the name of the field from the published attribute of the <cics:fieldnames> element in the Atom configuration

file for the Atom feed. If CICS passes this information to the service routine, the service routine can use this named field to locate the value of the timestamp or ABSTIME value that can be used to construct the value returned in the **ATMP_PUBLISHED** parameter. This data is used in the <atom:published> element for the entry.

ATMP_UPDATED_FLD
The name of a field in your resource records that contains the time when the resource was last updated. CICS obtains the name of the field from the updated attribute of the <cics:fieldnames> element in the Atom configuration file for the Atom feed. If CICS passes this information to the service routine, the service routine can use this named field to locate the value of the timestamp or ABSTIME value that can be used to construct the value returned in the **ATMP_UPDATED** parameter. This data is used in the <atom:updated> element for the entry.

ATMP_EDITED_FLD
The name of a field in your resource records that contains the time when the resource was last edited. CICS obtains the name of the field from the edited attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the value of the timestamp or ABSTIME value that can be used to construct the value returned in the **ATMP_EDITED** parameter. This data is used in the <app:edited> element for the entry.

ATMP_TITLE_FLD
The name of a field in your resource records that contains the title of the requested Atom entry. CICS obtains the name of the field from the title attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the title for the entry and return it in the DFHATOMTITLE container. The data from the DFHATOMTITLE container is used in the <atom:title> element for the entry.

ATMP_SUMMARY_FLD
The name of a field in your resource records that contains the summary of the requested Atom entry. CICS obtains the name of the field from the summary attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the summary for the entry and return it in the DFHATOMSUMMARY container. The data from the DFHATOMSUMMARY container is used in the <atom:summary> element for the entry.

ATMP_CONTENT_FLD
The name of a field in your resource records that contains the whole content of the requested Atom entry. CICS obtains the name of the field from the content attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the content for the entry and return it in the DFHATOMCONTENT container. The data from the DFHATOMCONTENT container is used in the <atom:content> element for the entry.

ATMP_CONTENT_TYPE_FLD
The name of a field in your resource records that contains the media type for the content of the Atom entry, such as application/xml or text. As for the **ATMP_MTYPEOUT** parameter, the media types "text", "html", and "xhtml" are used in place of the full IANA media types. The media type is specified in the type attribute of the <atom:content> element for an Atom entry. CICS obtains the name of the field from the content_type attribute of the <cics:fieldnames>

element. If CICS passes this information to the service routine, the service routine can use this named field to locate the media type for the content and determine suitable markup for the content in the DFHATOMCONTENT container. If the resource records do not have a field to store the media type for the content of the Atom entry, the media type specified in the type attribute of the <atom:content> element in the Atom configuration file applies. CICS passes this media type to the service routine in the **ATMP_MTYPEOUT** parameter.

ATMP_CATEGORY_FLD
The name of a field in your resource records that contains a category term that applies to the requested Atom entry. CICS obtains the name of the field from the category attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the category and return it in the DFHATOMCATEGORY container. The data from the DFHATOMCATEGORY container is used in the <atom:category> element for the entry.

ATMP_AUTHOR_FLD
The name of a field in your resource records that contains the name of the principal author of the Atom entry. CICS obtains the name of the field from the author attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the author's name and return it in the DFHATOMAUTHOR container. The data from the DFHATOMAUTHOR container is used in the <atom:name> element for the entry.

ATMP_AUTHORURI_FLD
The name of a field in your resource records that contains the URI of a Web site associated with the principal author of the Atom entry. CICS obtains the name of the field from the authoruri attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the URI and return it in the DFHATOMAUTHORURI container. The data from the DFHATOMAUTHORURI container is used in the <atom:uri> element for the entry.

ATMP_EMAIL_FLD
The name of a field in your resource records that contains the email address of the principal author of the Atom entry. CICS obtains the name of the field from the email attribute of the <cics:fieldnames> element. If CICS passes this information to the service routine, the service routine can use this named field to locate the email address and return it in the DFHATOMEMAIL container. The data from the DFHATOMEMAIL container is used in the <atom:email> element for the entry.

Input-output parameters in DFHATOMPARMS container

The service routine uses these parameters to supply information to CICS about the Atom entry that is being returned.

Each of the input-output parameters in the DFHATOMPARMS container is the address of a triple word containing a pointer to an area, the current length of the data in the area, and the maximum length of the area.

To supply information to CICS using a parameter, the service routine can do either of the following:

- Copy some data into the area indicated by the pointer, and set the current length of the area to the length of the data. The storage for the values of the

input-output parameters in the DFHATOMPARMS container is in user key, so you can access it when the service routine is defined with EXECKEY(USER).

- Set the pointer to some data in the service routine's own storage, which must last beyond the lifetime of the program (such as TWA storage), and set the current length of the area to the length of the data. You might need to do this if you have a value that is longer than the maximum length of the area provided.

If the service routine has no information relating to a particular parameter and CICS must use the default that it provides for the parameter, the service routine must indicate this to CICS by setting the current length of the data to zero.

ATMP_ATOMID
The Atom ID for the entry. An Atom ID is a unique identifier for the Atom entry. For more information about the format of Atom IDs, see "Atom IDs for Atom entries" on page 255.

On input, CICS uses this area to send the prototype Atom ID for the entry to the service routine. You determine the format of the prototype Atom ID by including either the <cics:authority> element or the <atom:id> element in the Atom configuration file, depending on whether you want to use the tag URI format or an alternative format to produce a unique identifier. CICS ignores Atom IDs that are supplied by Web clients, and does not pass these to the service routine.

The Atom format specification in RFC 4287 recommends that you store an Atom ID in the resource record for the Atom entry. For a POST request, if your resource can store Atom IDs, your service routine must complete the prototype Atom ID by appending the selector value for the Atom entry, as specified in the **ATMP_SELECTOR** parameter, and then store the complete Atom ID in the appropriate field in the resource record, as specified in the **ATMP_ID_FLD** parameter. If you prefer, your service routine can ignore the prototype Atom ID and substitute its own valid Atom ID for the Atom entry. The service routine can use the values of the **ATMP_TAG_AUTHORITY** and **ATMP_TAG_DATE** parameters as input to construct an Atom ID.

Note that if you are using the tag URI format, the resulting Atom ID is unique for Atom entries provided by a user-written service routine that deals with a single Atom feed, but it is not unique if the user-written service routine provides more than one feed. If your user-written service routine provides multiple feeds, either choose an alternative format for your Atom IDs, or use a different authority name or date in the <cics:authority> element of the Atom configuration file for each of the feeds.

On output, the service routine must use the **ATMP_ATOMID** parameter as follows:

- If you have stored a complete Atom ID in the resource record for the Atom entry, the service routine must return the complete Atom ID from the field in the resource record, as specified in the **ATMP_ID_FLD** parameter, followed by the length of the Atom ID.
- If your resource does not store Atom IDs, the service routine must set the current length of the data for the **ATMP_ATOMID** parameter to zero. In this case, CICS appends the selector value to the prototype Atom ID to produce a complete Atom ID.

ATMP_ETAGVAL
An entity tag (or Etag) value for the selected resource record. To produce an entity tag, a service routine can use the EXEC CICS BIF DIGEST command to calculate the SHA-1 digest of the record, or use another suitable method to produce an entity tag that complies with the HTTP/1.1 protocol requirements.

On input, CICS uses the **ATMP_ETAGVAL** parameter to provide any entity tag for the Atom entry to the service routine. When a Web client makes a PUT or DELETE request to edit an Atom entry, CICS requires the client to supply an If-Match HTTP header on the request containing an entity tag. If CICS provides an entity tag using this parameter, the service routine must calculate the entity tag for the existing record and compare it to the Web client's entity tag. If the tags do not match, indicating that the entry has been changed by another agent, the service routine must reject the request with the response code `atmp_resp_etag_no_match`. A Web client might supply an asterisk in place of an entity tag to indicate that the entry should be edited or deleted even if it has been changed by another agent, and the service routine should comply with this request.

On output, the service routine must use the **ATMP_ETAGVAL** parameter as follows:

- Entity tags are not used for entries in an Atom feed. If the current Atom entry is part of an Atom feed, the service routine must set the current length of the data to zero.
- CICS requires entity tags for entries in a collection. If the current Atom entry is part of a collection, the service routine must calculate and return the entity tag. Do not store entity tags in resource records; calculate them when they are needed.

ATMP_PUBLISHED
The service routine can use this parameter to return the date and time at which the returned Atom entry was first published. "Published" means the point when the data was first created or first made available. If your resource does not store this data, the service routine must indicate this by setting the current length of the data to zero, and in this case CICS provides the default of the current time. If the service routine returns a date and time stamp, it must be in the RFC 3339 format, also known as the XML dateTime datatype. You can use the EXEC CICS FORMATTIME command to provide a date and time stamp in this format, or if your service routine can use the TRANSFORM DATATOXML command, you can convert a CICS ABSTIME value into a date and time stamp in this format.

ATMP_UPDATED
The service routine can use this parameter to return the date and time at which the returned Atom entry was last updated. "Updated" means a point when the data was changed in a way that you consider to be significant. If your resource does not store this data, the service routine must indicate this by setting the current length of the data to zero, and in this case CICS provides the default of the current time. If the service routine returns a date and time stamp, it must be in the RFC 3339 format.

ATMP_EDITED
The service routine can use this parameter to return the date and time at which the returned Atom entry was last edited. If your resource does not store this data, the service routine must indicate this by setting the current length of the data to zero, and in this case CICS provides the default of the current time. If the service routine returns a date and time stamp, it must be in the RFC 3339 format.

ATMP_SELECTOR
A selector value for the Atom entry that the service routine must provide. “Selector value for Atom entries” on page 251 explains what a selector value is.

- When a client is making a general request for a feed, on input, CICS sends a null value for the **ATMP_SELECTOR** parameter, and the input parameter

ATMP_ATOMTYPE has the value "feed". On receiving this combination of values, the service routine must take the following actions:

- Return the data for the most recent Atom entry that was added to the feed.
- Use the **ATMP_SELECTOR** parameter to return a selector value identifying that entry. If your resource does not hold Atom IDs for entries, CICS uses this selector value in the generated Atom ID for the entry.
- Use the **ATMP_NEXTSEL** parameter to return a selector value for the next entry in the feed.

- When CICS needs a second or subsequent Atom entry from a feed to complete a client request, or a client has requested a feed document containing a specific range of Atom entries, on input, CICS uses the **ATMP_SELECTOR** parameter to send a selector value for one of the Atom entries in the feed document, and the input parameter **ATMP_ATOMTYPE** has the value "feed". On receiving this combination of values, the service routine must take the following actions:
  - Return the data for the Atom entry that is represented by the selector value.
  - Do not change the data or length that CICS supplied for the **ATMP_SELECTOR** parameter.
  - Use the **ATMP_NEXTSEL** parameter to return a selector value for the next entry in the feed.
- When a client is requesting a specific entry from a feed, CICS uses the **ATMP_SELECTOR** parameter to send the selector value extracted from the URL for the entry, and the input parameter **ATMP_ATOMTYPE** has the value "entry". On receiving this combination of values, the service routine must take the following actions:
  - Return the data for the Atom entry that is represented by the selector value.
  - Do not change the data or length that CICS supplied for the **ATMP_SELECTOR** parameter.
  - For the **ATMP_NEXTSEL** parameter, return a null value by setting the current length of the data to zero.

Note: For a collection, CICS uses the value "collection" for the **ATMP_ATOMTYPE** parameter in the situations where the value "feed" would be used for an Atom feed. The value "entry" is the same for an entry from a collection or an entry from an Atom feed.

ATMP_NEXTSEL
The service routine must use this parameter to return a selector value for the next Atom entry that is available, if any. “Sequence for Atom entries” on page 252 explains the order in which you should return your Atom entries.

This value must be supplied whether the service routine is handling a feed or a collection. It is not required when the client requests a single specific entry (with the value "entry" for **ATMP_ATOMTYPE**), or when no more data is available to provide Atom entries. When the value is not required, the service routine must return a null value for this parameter by setting the current length of the data to zero.

CICS uses the value supplied by the service routine to request further Atom entries from the service routine to complete the Atom document. If the Atom document is complete, CICS uses this value to produce the <atom:link

rel="next"> link in the Atom document, which Web clients can use to retrieve the next window of Atom entries from the feed or the next partial list of Atom entries from the collection.

ATMP_PREVSEL
A service routine that is handling a collection must use this parameter to return a selector value for the previous Atom entry in the collection, if any. “Sequence for Atom entries” on page 252 explains the order in which you should return your Atom entries.

This value must be supplied when the **ATMP_ATOMTYPE** parameter has the value "collection". It is not required when the client requests a single specific entry (with the value "entry" for **ATMP_ATOMTYPE**), or when there is no previous Atom entry. When the value is not required, the service routine must return a null value for this parameter by setting the current length of the data to zero.

When the Atom document is complete, CICS uses this value to carry out a chain of requests to the service routine to produce the <atom:link rel="previous"> link in the Atom document, which Web clients can use to retrieve the previous partial list of Atom entries from the collection.

This value is not required from a service routine that is handling an ordinary Atom feed. You may specify it if the <atom:link rel="previous"> link would be useful to your Web clients in order to retrieve the previous window of Atom entries from the feed. However, the processing to produce this link increases response times, so only specify this value for a feed if your Web clients are set up to use this form of navigation.

ATMP_FIRSTSEL
A service routine that is handling a collection must use this parameter to return a selector value for the first Atom entry in the collection. “Selector value for Atom entries” on page 251 explains the order in which you should return your Atom entries.

This value must be supplied when the **ATMP_ATOMTYPE** parameter has the value "collection". On subsequent calls relating to the same Web client request, CICS uses the **ATMP_FIRSTSEL** parameter to supply this selector value to the service routine, so the service routine does not need to provide it again.

The value is not required when the client requests a single specific entry (with the value "entry" for **ATMP_ATOMTYPE**). When the value is not required, the service routine must return a null value for this parameter by setting the current length of the data to zero.

When the Atom document is complete, CICS uses this value to produce the <atom:link rel="first"> link in the Atom document, which Web clients can use to retrieve the first (newest) partial list of Atom entries from the collection.

This value is not required from a service routine that is handling an ordinary Atom feed. You may specify it if the <atom:link rel="first"> link would be useful to your Web clients in order to retrieve the first (newest) window of Atom entries from the feed. CICS does not carry out any additional processing to produce this link.

ATMP_LASTSEL
A service routine that is handling a collection must use this parameter to return a selector value for the last Atom entry in the collection. “Sequence for Atom entries” on page 252 explains the order in which you should return your Atom entries.

This value must be supplied when the **ATMP_ATOMTYPE** parameter has the value "collection". On subsequent calls relating to the same Web client request, CICS uses the **ATMP_LASTSEL** parameter to supply this selector value to the service routine, so the service routine does not need to provide it again.

The value is not required when the client requests a single specific entry (with the value "entry" for **ATMP_ATOMTYPE**). When the value is not required, the service routine must return a null value for this parameter by setting the current length of the data to zero.

When the Atom document is complete, CICS uses this value to produce the <atom:link rel="last"> link in the Atom document, which Web clients can use to retrieve the last (oldest) partial list of Atom entries from the collection. CICS issues this last partial list containing only a single entry, that is, the last entry in the feed. Web clients can use the <atom:link rel="previous"> links to retrieve all the previous partial lists.

This value is not required from a service routine that is handling an ordinary Atom feed. You may specify it if the <atom:link rel="last"> link would be useful to your Web clients in order to retrieve the last (oldest) Atom entry from the feed. CICS does not carry out any additional processing to produce this link.

ATMP_OPTIONS parameter in DFHATOMPARMS container

The **ATMP_OPTIONS** parameter is the address of a double word containing 64 option bits that you use to indicate that your service routine is supplying optional containers with data such as a title for the Atom entry. The options string is mapped by the **ATMP_OPTIONS_BITS** DSECT.

The options bitmap to which **ATMP_OPTIONS** points is mapped in two ways: ATMP_OPTIONS_BITS and ATMP_OPTIONS_WORDS. ATMP_OPTIONS_BITS is a series of byte and bit definitions for use in languages that understand bit values. ATMP_OPTIONS_WORDS is a pair of fullword values, for use in COBOL, where bit values cannot be easily coded.

ATMP_OPTIONS_BITS

In ATMP_OPTIONS_BITS, the bit values that have meaning are in byte ATMP_OUTOPT_BYTE1, and they are as follows:

OPTTITLE
The service routine is using the DFHATOMTITLE container to return a character string to be used as the title for the entry.

OPTSUMMA
The service routine is using the DFHATOMSUMMARY container to return a character string to be used as the summary for the entry.

OPTAUTHOR
The service routine is using the DFHATOMAUTHOR container to return a character string to be used as the name of the author of the entry.

OPTAUTHEML
The service routine is using the DFHATOMEMAIL container to return a character string to be used as the e-mail address for the author of the entry.

OPTAUTHURI
The service routine is using the DFHATOMAUTHORURI container to return a character string to be used as the URI of a Web site associated with the author of the entry.

OPTCATEG
The service routine is using the DFHATOMCATEGORY container to return a character string to be used as a category term for the entry.

ATMP_OPTIONS_WORDS

ATMP_OPTIONS_WORDS contains these two fullword values:

ATMP_OPTIONS_IN
A fullword of input option values, which is not used.

ATMP_OPTIONS_OUT
A fullword in which to store output option values. The fullword values equivalent to the bit values in ATMP_OUTOPT_BYTE1 are specified in copybook DFH0W2CO. These values can be added together, as required, to produce a suitable bitmap value.

Copybook DFH0W2CO contains binary integers representing the value of the bits in ATMP_OUTOPT_BYTE1, for use in ATMP_OPTIONS_OUT, as follows:

OPTTITLE_NUM
Equivalent to OPTTITLE

OPTSUMMA_NUM
Equivalent to OPTSUMMA

OPTAUTHOR_NUM
Equivalent to OPTAUTHOR

OPTAUTHEML_NUM
Equivalent to OPTAUTHEML

OPTCATEG_NUM
Equivalent to OPTCATEG

OPTAUTHURI_NUM
Equivalent to OPTAUTHURI

ATMP_RESPONSE parameter in DFHATOMPARMS container

The **ATMP_RESPONSE** parameter is the address of a double word that you use to return a response code to CICS indicating success or error. If the service routine sends a response code indicating an error, CICS produces a suitable default HTTP error response to send to the Web client. The service routine can use the DFHHTTPSTATUS container to return an alternative status code and text to override the default error response.

Do not change the address of the double word. The first fullword, **ATMP_RESPONSE_CODE**, contains the response code. It is initialized to zero, indicating successful completion. The second fullword, **ATMP_REASON_CODE**, is also initialized to zero, and the service routine must not change this fullword; it is reserved for future use.

The symbol values for **ATMP_RESPONSE_CODE** are defined in the DFHW2CN series of copybooks. The values are as follows:

```
atmp_resp_normal           constant(0);  ! Normal success response
atmp_resp_not_found        constant(4);  ! Resource not found
atmp_resp_not_auth         constant(8);  ! Resource not authorized
atmp_resp_disabled         constant(12); ! Resource is disabled
atmp_resp_already_exists   constant(16); ! Resource already exists
atmp_resp_etag_no_match    constant(20); ! If-Match compare failed
```

```
atmp_resp_invalid_request   constant(24); ! Request not valid
atmp_resp_access_error      constant(32); ! Other resource error
atmp_resp_conversion_failed constant(36); ! XML Conversion error
```

If the parameter is returned unchanged, CICS sends an HTTP response indicating successful completion of the request. If the service routine sends a response code indicating an error, CICS produces a suitable default HTTP error response to send to the Web client. The default HTTP error responses are as follows:

Table 12. Default HTTP error responses from service routines for Atom feeds

ATMP_RESPONSE_CODE value	**HTTP status code**	**HTTP status text**
atmp_resp_normal	200 (201 for POST requests)	OK (For POST requests, Created)
atmp_resp_not_found	404	Not found
atmp_resp_not_auth	403	Forbidden
atmp_resp_disabled	503	Service unavailable
atmp_resp_already_exists	409	Duplicate resource
atmp_resp_etag_no_match	412	Precondition failed
atmp_resp_invalid_request	400	Invalid request
atmp_resp_access_error	500	Resource error
atmp_resp_conversion_failed	500	Resource error

When the service routine returns an error, it can use the DFHHTTPSTATUS container to return an alternative status code and text to replace the default HTTP error response. For a listing of status codes that you might want to use in your error responses, see Appendix C, “HTTP status code reference for CICS Web support,” on page 381. You cannot override the default HTTP response for a successful request (with a zero response code).

DFHATOMCONTENT container

DFHATOMCONTENT is a container of DATATYPE(CHAR) that you use for your service routine to provide the content for an Atom entry.

This container is required. CICS returns the data in the container as the <atom:content> element for the Atom entry.

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_CONTENT_FLD** parameter has the name and length of the field in your resource record that holds the data for this container. You can use data from a single field or from a structure of nested fields. If you are not using the resource handling parameters, you can code your service routine either to return the whole of the resource record as the content for the entry, or to select appropriate fields from the resource record to assemble the content.

You can supply your content as plain text with no child elements, or you can use XML or another type of markup, such as HTML or XHTML, to format your data. The **ATMP_MTYPEOUT** parameter in the DFHATOMPARMS container contains the media type for the expected content of the Atom entry, as specified in the type attribute of the <atom:content> element in the Atom configuration file for the Atom feed. As in RFC 4287, the media type "text" is used for plain text instead of the IANA media type "text/plain", "html" is used instead of "text/html", and

"xhtml" is used instead of "application/xhtml+xml". If you do not specify a media type in the Atom configuration file, CICS supplies a default media type of "application/xml" in this parameter.

You can also store media types for individual Atom entries in your resource records. If you are using the resource handling parameters, the **ATMP_CONTENT_TYPE_FLD** parameter has the name and length of the field containing the media type for the content of the Atom entry.

<content> tags for Atom entry content

If your content is *not* in the expected media type that CICS supplied to the service routine in the **ATMP_MTYPEOUT** parameter in the DFHATOMPARMS container, you must include the tag <content> in the container at the beginning of your content, and the closing tag </content> at the end. If your content is anything other than plain text, you must also add a type attribute to the <content> tag to specify the media type for your content. Some possible type attributes are as follows:

- <content type="html"> specifies HTML.
- <content type="xhtml"> specifies XHTML.
- <content type="text/xml"> is the media type that is normally used for a human-readable XML document.

You may specify <content type="text"> when you supply plain text, but the recipients of Atom documents assume this media type if you do not specify any type attribute. If your content is in any other format, specify the IANA media type that you would normally use for that format on the Internet. A listing of media types is available at http://www.iana.org/assignments/media-types/. Note that CICS does not provide support for nontext media types.

For content that is in the expected media type that CICS supplied to the service routine in the **ATMP_MTYPEOUT** parameter in the DFHATOMPARMS container, you can omit the <content> and </content> tags. In this case, CICS supplies the opening and closing tags and specifies the type attribute as the media type in the **ATMP_MTYPEOUT** parameter.

Markup for Atom entry content

If you use a format other than plain text for your content, read the processing information in Section 4.1.3.3 of RFC 4287, *The Atom Syndication Format*, which is available from http://www.ietf.org/rfc/rfc4287.txt. These rules explain how you must arrange your markup and how the recipients of Atom documents (who are known as "Atom Processors") interpret and present the content depending on the type of markup used. In particular, note that HTML markup must be escaped, for example, the tag "
" must be written as "
". CICS does not validate the markup that you use.

If you want to produce XML content from the fields in your resource record, and your resource has an XML binding with an associated XMLTRANSFORM resource, you can use the CICS functions for transforming application data into XML. If you have a language structure, or copybook, that describes the structure of the data in your resource in any one of the high-level languages supported by the DFHLS2SC procedure, that is, COBOL, C, C++, or PL/I, you can create an XML binding. Generate mappings from language structures explains how to do this. CICS dynamically creates the XMLTRANSFORM resource when you install an ATOMSERVICE resource definition that names the XML binding.

If an XMLTRANSFORM resource is available, CICS provides its name in the **ATMP_XMLTRANSFORM** parameter in the DFHATOMPARMS container. For more information about the TRANSFORM DATATOXML command and instructions for using the data mapping functions, see the *CICS Application Programming Guide*.

Returning Atom entry metadata in containers

If the resource that contains the data for your Atom entries has fields in its records that supply metadata for the individual Atom entries, such as a title or a summary, use the optional metadata containers, such as DFHATOMTITLE, for your service routine to provide this data to CICS.

About this task

The DFHATOMCONTENT container, which holds the content for the Atom entry, is the only container that the service routine is required to return to CICS. A number of other optional containers are available to return any metadata that your service routine can extract from the records in the resource that contains the data for the Atom entries. The records in your resource might not contain any fields to hold metadata, for example, if you are creating an Atom feed from an existing file that was not originally set up for use as an Atom feed. If you do not have any metadata in your records, you do not have to return these containers, but in some cases you will need to supply default metadata when you set up the Atom configuration file for the Atom feed.

The sample service routine DFH$W2S1 shows you how to create the optional containers using the PUT CONTAINER command, and how to populate them with data that you extracted from a record in your resource.

Procedure

1. If the records in your resource contain individual titles for Atom entries, use the PUT CONTAINER command to create a container named DFHATOMTITLE, with DATATYPE(CHAR), that contains the title of this Atom entry. This container is optional, but if you do not provide this data from your service routine, you must specify a default title in the Atom configuration file. “DFHATOMTITLE container” on page 285 explains what to put in the container.
2. If the entries in your resource have individual summaries, use the PUT CONTAINER command to create a container named DFHATOMSUMMARY, with DATATYPE(CHAR), that contains the summary for the Atom entry. This container is optional, but the Atom specification requires a summary if the content of an entry is not text or XML. “DFHATOMSUMMARY container” on page 285 explains what to put in the container.
3. If you want to use your service routine to provide the name of the author of the entry, use the PUT CONTAINER command to create a container named DFHATOMAUTHOR, with DATATYPE(CHAR), that contains the author's name. This container is optional, but if you do not provide this data from your service routine, you must specify a default name in the Atom configuration file or accept the CICS default. “DFHATOMAUTHOR container” on page 286 explains what to put in the container.
4. If you want to use your service routine to provide an e-mail address and URI (Web site address) for the author of the entry, use the PUT CONTAINER command to create containers named DFHATOMEMAIL and DFHATOMAUTHORURI, with DATATYPE(CHAR), that contain these items of data. These containers are optional, and you can provide either, both, or neither

of them. “DFHATOMEMAIL container” on page 286 and “DFHATOMURI container” on page 287 explain what to put in these containers.

5. If you want to use your service routine to provide a category for the entry, use the PUT CONTAINER command to create a container named DFHATOMCATEGORY, with DATATYPE(CHAR), that contains a category term for the entry. This container is optional.“DFHATOMCATEGORY container” on page 287 explains what to put in the container.
6. In the DFHATOMPARMS container, set the appropriate option bit from the **ATMP_OPTIONS_OUT** parameter for each optional container that you are returning to CICS. “DFHATOMPARMS container” on page 271 documents this parameter. The sample service routine DFH$W2S1 shows you how to set these option bits.

DFHATOMTITLE container

DFHATOMTITLE is a container of DATATYPE(CHAR) that you may use for your service routine to provide a title for an Atom entry.

This container is optional. CICS returns the data in the container as the <atom:title> element for the Atom entry.

If your resource records do not hold individual titles for Atom entries, you can use the Atom configuration file to specify the same title for every entry in your feed. Atom entries must have a title, so CICS requires a default title in the Atom configuration file. If there is no suitable default title, you can use a blank default title, but in this situation you must use the DFHATOMTITLE container to provide a title for every entry, in order to be compliant with the Atom format specification.

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_TITLE_FLD** parameter has the name and length of the field in your resource record that holds the data for this container.

Do not put tags around the data in the container, and do not use any markup to format it. CICS only supports plain text for titles.

When you provide this container to CICS, set the **OPTTITLE** bit value in the **ATMP_OPTIONS** parameter in the DFHATOMPARMS container.

DFHATOMSUMMARY container

DFHATOMSUMMARY is a container of DATATYPE(CHAR) that you may use for your service routine to provide a summary for an Atom entry.

This container is optional. CICS returns the data in the container as the <atom:summary> element for the Atom entry.

If your resource records do not hold individual summaries for Atom entries, you can use the Atom configuration file to specify the same summary for every entry in your feed. The Atom specifications require a summary if the content of an entry is not text or XML, so if you expect to provide content that does not fit these categories, you must either provide the DFHATOMSUMMARY container or use the Atom configuration file to provide this data. CICS does not check that you have provided a summary for a nontext and non-XML entry, so it is your responsibility to comply with the specification in this respect.

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_SUMMARY_FLD** parameter has the name and length of the field in your resource record that holds the data for this container.

Do not put tags around the data in the container, and do not use any markup to format it.

When you provide this container to CICS, set the **OPTSUMMA** bit value in the **ATMP_OPTIONS** parameter in the DFHATOMPARMS container.

DFHATOMAUTHOR container

DFHATOMAUTHOR is a container of DATATYPE(CHAR) that you may use for your service routine to provide the name of the author of an Atom entry.

This container is optional. CICS returns the data in the container as the <atom:name> child element of the <atom:author> element for the Atom entry.

The Atom specification requires an author name for each Atom entry. If the records in your resource have a field in which individual authors are named, provide this data; otherwise your service routine can provide the name of the same author for every entry in your feed. As an alternative, you can use the Atom configuration file for the feed to provide the name of a single author for every entry in your feed. If you do not provide an author's name by any means, CICS will send the response to the Web client, but with a default author name of "CICS Transaction Server".

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_AUTHOR_FLD** parameter has the name and length of the field in your resource record that holds the data for this container.

Do not put tags around the data in the container, and do not use any markup to format it.

When you provide this container to CICS, set the **OPTAUTHOR** bit value in the **ATMP_OPTIONS** parameter in the DFHATOMPARMS container.

DFHATOMEMAIL container

DFHATOMEMAIL is a container of DATATYPE(CHAR) that you may use for your service routine to provide an e-mail address for the author of an Atom entry.

This container is optional. CICS returns the data in the container as the <atom:email> child element of the <atom:author> element for the Atom entry.

The Atom specification does not require this data, so you can omit it if you do not have it or if you do not want to distribute it. If the records in your resource include e-mail addresses for individual authors, you can provide this data. If you are providing the name of a single author for every entry in your feed, either through the service routine or in the Atom configuration file for the feed, you can do the same for the email address.

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_EMAIL_FLD** parameter has the name and length of the field in your resource record that holds the data for this container.

Do not put tags around the data in the container, and do not use any markup to format it.

When you provide this container to CICS, set the **OPTEMAIL** bit value in the **ATMP_OPTIONS** parameter in the DFHATOMPARMS container.

DFHATOMURI container

DFHATOMURI is a container of DATATYPE(CHAR) that you may use for your service routine to provide a URI (Web site address) that is relevant to the author of an Atom entry.

This container is optional. CICS returns the data in the container as the <atom:uri> child element of the <atom:author> element for the Atom entry.

As for the author's e-mail address, the Atom specification does not require this data, so you can omit it if you do not have it or if you do not want to distribute it. If the records in your resource include Web sites for individual authors, you can provide this data. If all the authors are from your company, you could supply the URI of the home page for your company. If you are providing the name of a single author for every entry in your feed, either through the service routine or in the Atom configuration file for the feed, you can do the same for the URI.

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_AUTHORURI_FLD** parameter has the name and length of the field in your resource record that holds the data for this container.

Do not put tags around the data in the container, and do not use any markup to format it.

When you provide this container to CICS, set the **OPTAUTHURI** bit value in the **ATMP_OPTIONS** parameter in the DFHATOMPARMS container.

DFHATOMCATEGORY container

DFHATOMCATEGORY is a container of DATATYPE(CHAR) that you may use for your service routine to provide a category for an Atom entry.

This container is optional. CICS returns the data in the container as the term attribute of an <atom:category> element for the Atom entry. CICS does not support the optional scheme and label attributes for the <atom:category> element.

If your resource records do not include categories for Atom entries, you can use the Atom configuration file to specify the same category for every entry in your feed, if you know a reason why this would be helpful. The Atom specifications do not require categories.

If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_CATEGORY_FLD** parameter has the name and length of the field in your resource record that holds the data for this container.

Do not put tags around the data in the container, and do not use any markup to format it.

When you provide this container to CICS, set the **OPTCATEG** bit value in the **ATMP_OPTIONS** parameter in the DFHATOMPARMS container.

DFH$W2S1 C sample service routine for Atom feeds

The sample service routine DFH$W2S1 is a skeleton program in C language that shows you how to read the parameters in the DFHATOMPARMS container, update the metadata and content containers (such as DFHATOMTITLE and DFHATOMCONTENT), and update and return the DFHATOMPARMS container.

If you create and install an appropriate RDO definition, which must specify EXECKEY(CICS), for DFH$W2S1, you can run the sample service routine to supply some data for test purposes in response to GET requests for an Atom feed. As shipped, the program returns only the default data that is set up by its code, and it does not handle POST, PUT, or DELETE requests. The process of identifying the required record from the resource that holds the data for your Atom entries, and extracting the appropriate fields from the record, or updating the record in response to a POST, PUT, or DELETE request, is specific to your choice of resource and the structure of the records in that resource. For an example of how your service routine can interact with your resource, see the description of DFH0W2F1 in "DFH0W2F1 COBOL sample service routine for Atom feeds" on page 353. DFH0W2F1 interacts with the CICS sample file FILEA.

The DFH$W2S1 sample service routine performs the following tasks:

- Includes the C header file for the copybook DFHW2APH that contains the parameter list for DFHATOMPARMS, and the copybook DFHW2CNH that contains the constants for the response codes.
- Defines a structure called "outdata" that will be placed in the temporary work area (TWA). The program will use this structure to store new data for the DFHATOMPARMS parameters, and will return pointers to the new data for CICS to replace the existing values of the parameters. The storage for your new data for the DFHATOMPARMS parameters must be in the TWA so that it can be read by CICS Atom processing.

```
typedef struct
{
    char atomid??(50??);
    char published??(30??);
    ...
    char selector??(20??);
} outdata;
```

- Defines a structure called "indata", which is a placeholder that shows you how to extract all the values from the DFHATOMPARMS parameters. In practice, your service routine can extract the values as it needs them to perform each processing step.

```
typedef struct
{
    char* resname;
    char* restype;
    char* atomtype;
    ...
    char* emailfld;
} indata;
```

- Provides a helper method getParameter that you can use to obtain the pointer and length information from a parameter in DFHATOMPARMS, read the value of the parameter into memory, add a null terminator (zero byte) so that it can be used as a C string, and return a pointer to the new memory location.

```
char* getParameter(atmp_parameter* ParamPtr)
{
    char* DataPtr;
    int DataLen;
    DataLen  = ParamPtr->atmp_parameter_len;

    EXEC CICS GETMAIN SET(DataPtr) FLENGTH(DataLen+1)
                      INITIMG(0x00);
    if (DataLen != 0) {
        memcpy(DataPtr, ParamPtr->atmp_parameter_ptr, DataLen);
```

```
    }

    return DataPtr;
}
```

- Provides a helper method updatedAtomParam that you can use to update the pointer and length that CICS sent for a DFHATOMPARMS parameter with a pointer to a new string value containing your new data, and the length of that string.

```
void updatedAtomParam(char *value, atmp_parameter *ParamPtr)
{
    ParamPtr->atmp_parameter_ptr = (unsigned long*)value;
    ParamPtr->atmp_parameter_len = strlen(value);
}
```

- Provides a helper method updateAtomContainer that uses the EXEC CICS PUT CONTAINER command to update one of the metadata and content containers (such as DFHATOMTITLE and DFHATOMCONTENT) with data for your Atom entry, taken from a field in the record in the resource that holds the data for your Atom entries, and set the corresponding option bit to which DFHATOMPARMS supplied a pointer.

```
void updateAtomContainer(char *cName, char *cChannel, char *value,
                         atmp_options_bits *optPtr)
{
    int len = strlen(value);

    EXEC CICS PUT CONTAINER(cName)
                  CHANNEL(cChannel)
                  FROM(value)
                  FLENGTH(len)
                  FROMCODEPAGE("IBM037");

    /* work out which option to set */
    if (strcmp(cName, "DFHATOMTITLE    ") == 0) {
        (*optPtr).atmp_options_outbit.atmp_outopt_byte1.opttitle = 1;
    }
    else if (strcmp(cName, "DFHATOMSUMMARY  ") == 0) {
        (*optPtr).atmp_options_outbit.atmp_outopt_byte1.optsumma = 1;
    }
    else if (strcmp(cName, "DFHATOMCATEGORY  ") == 0) {
        (*optPtr).atmp_options_outbit.atmp_outopt_byte1.optcateg = 1;
    }
    else if (strcmp(cName, "DFHATOMAUTHOR    ") == 0) {
        (*optPtr).atmp_options_outbit.atmp_outopt_byte1.optauthor = 1;
    }
    else if (strcmp(cName, "DFHATOMAUTHORURI ") == 0) {
        (*optPtr).atmp_options_outbit.atmp_outopt_byte1.optautheml = 1;
    }
    else if (strcmp(cName, "DFHATOMEMAIL     ") == 0) {
        (*optPtr).atmp_options_outbit.atmp_outopt_byte1.optauthuri = 1;
    }

}
```

- In the main program, sets up variables such as the storage for data that will be placed into the metadata and content containers (such as DFHATOMTITLE and DFHATOMCONTENT), and the "outdata" and "indata" structures.
- Issues an EXEC CICS ADDRESS TWA command to set up storage for the "outdata" structure in the TWA, and populates this structure with default values. These default values are returned if you run the sample service routine as supplied. By setting the value for the **ATMP_NEXTSEL** parameter to 0, the sample service routine makes CICS request the same Atom entry, consisting of this default data, over and over until the number of entries specified as the window for the Atom feed document are obtained. When you code your own service

routine that supplies real data, do not use default values, because depending on the request method, you might want to leave a number of data items unchanged.

```
EXEC CICS ADDRESS TWA(outData);

    strcpy(outData->atomid,
           "cics-atomservice-sample:2009-02-02T00:00:00Z");
    strcpy(outData->published, "2009-02-02T00:00:00Z");
    strcpy(outData->updated, "2009-02-20T14:54:34Z");
    strcpy(outData->edited, "2009-02-20T14:54:34Z");
    strcpy(outData->etagval, "");
    strcpy(outData->selector, "");
    strcpy(outData->nextsel, "0");
    strcpy(outData->prevsel, "");
    strcpy(outData->firstsel, "");
    strcpy(outData->lastsel, "");
```

- Issues EXEC CICS GETMAIN commands to obtain storage to hold the values of the metadata and content containers (such as DFHATOMTITLE and DFHATOMCONTENT), and populates these containers with default data. The data in the containers does not need to be in the TWA. The content for the Atom entry, in the DFHATOMCONTENT container, must be enclosed in <atom:content> opening and closing tags, that may specify a type attribute giving the type of content. You may abbreviate <atom:content> to just <content>, as the sample service routine does.

```
    EXEC CICS GETMAIN SET(AtomContent)   FLENGTH(512) INITIMG(0x00);
    EXEC CICS GETMAIN SET(AtomTitle)     FLENGTH(50) INITIMG(0x00);
    EXEC CICS GETMAIN SET(AtomSummary)   FLENGTH(100) INITIMG(0x00);
    EXEC CICS GETMAIN SET(AtomCategory)  FLENGTH(30) INITIMG(0x00);
    EXEC CICS GETMAIN SET(AtomAuthor)    FLENGTH(40) INITIMG(0x00);
    EXEC CICS GETMAIN SET(AtomAuthorUri) FLENGTH(256) INITIMG(0x00);
    EXEC CICS GETMAIN SET(AtomEmail)     FLENGTH(256) INITIMG(0x00);

    strcpy(AtomContent,
           "<content type='text/xml'>Hello world</content>");
    strcpy(AtomTitle,     "Sample Service Routine Entry");
    strcpy(AtomSummary,
           "This is an entry from the sample service routine");
    strcpy(AtomCategory,  "sample-entry");
    strcpy(AtomAuthor,    "CICS Sample service routine");
    strcpy(AtomAuthorUri, "");
    strcpy(AtomEmail,     "");
```

- Issues an EXEC CICS ASSIGN CHANNEL command to get the name of the current channel for use on the later GET and PUT CONTAINER commands.

```
    EXEC CICS ASSIGN CHANNEL(cChannel);
```

- Checks for the presence of the DFHREQUEST container, which CICS uses to pass the body of a Web client's POST or PUT request to the service routine. The first EXEC CICS GET CONTAINER command requests the length of the container, and if a nonzero response code is returned, meaning that either the container does not exist or there is some problem with it, does not proceed. If a request body has been passed, the sample service routine reads its contents into storage.

```
    EXEC CICS GET CONTAINER("DFHREQUEST      ")
                  CHANNEL(cChannel)
                  NODATA
                  FLENGTH(DataLen)
                  RESP(Resp) RESP2(Resp2);

    if(Resp != 0 || Resp2 != 0)
    {
        DataLen = -1;
    }
```

```
/* If we have a request body then get it */
if(DataLen != -1)
{
    DataLen++;
    EXEC CICS GETMAIN SET(RequestBody) FLENGTH(DataLen)
                      INITIMG(0x00);

    EXEC CICS GET CONTAINER("DFHREQUEST      ")
                      INTO(RequestBody)
                      FLENGTH(DataLen);
}
```

- Issues an EXEC CICS GET CONTAINER command to set a pointer ParamList to the beginning of the data for the DFHATOMPARMS parameters.

```
EXEC CICS GET CONTAINER("DFHATOMPARMS    ")
             SET(ParamListData)
             FLENGTH(DataLen);

ParamList = (atmp_parameter_list*)ParamListData;
```

- Goes through the data for the DFHATOMPARMS parameters, and uses the helper method getParameter to obtain the value of each parameter and set a pointer for it in the "inData" structure.

```
optPtr = (atmp_options_bits*)ParamList->atmp_options;

ParamPtr = (atmp_parameter*)ParamList->atmp_resname;
inData.resname = getParameter(ParamPtr);

ParamPtr = (atmp_parameter*)ParamList->atmp_restype;
inData.restype = getParameter(ParamPtr);

...

ParamPtr = (atmp_parameter*)ParamList->atmp_email_fld;
inData.emailfld = getParameter(ParamPtr);
```

- Indicates where you must provide code that interacts with the resource that holds the data for your Atom entries, based on the information in the DFHATOMPARMS parameters and the DFHREQUEST container. Your service routine needs to locate the required record in the resource and extract the appropriate fields from the record in response to a GET request, or update the record in response to a POST, PUT, or DELETE request. The value of the **ATMP_HTTPMETH** parameter tells your service routine what the request method is.
- Shows you how to place the data that you have obtained from your resource record into the memory locations indicated by the pointers in the outData structure, and into the storage for the metadata and content containers (such as DFHATOMTITLE and DFHATOMCONTENT). The example shows you how to add a selector value for the current record if the **ATMP_SELECTOR** parameter was null on input for the service routine.

```
 *  strcpy(outData->atomid,    ".....");
 *  strcpy(outData->published, ".....");
 *  strcpy(outData->updated,   ".....");
 *  strcpy(outData->edited,    ".....");
 *  strcpy(outData->etagval,   ".....");
 *  strcpy(outData->nextsel,   ".....");
 *  strcpy(outData->prevsel,   ".....");
 *  strcpy(outData->firstsel,  ".....");
 *  strcpy(outData->lastsel,   ".....");
 *
 *  if (strlen(inData->selector) == 0) {
 *      strcpy(outData->selector, ".....");
 *  }
 *
```

```
 *  strcpy(AtomContent,   "....");
 *  strcpy(AtomTitle,     "....");
 *  strcpy(AtomSummary,   "....");
 *  strcpy(AtomCategory,  "....");
 *  strcpy(AtomAuthor,    "....");
 *  strcpy(AtomAuthorUri, "....");
 *  strcpy(AtomEmail,     "....");
```

- Uses the updatedAtomParam helper method to update the storage for each DFHATOMPARMS parameter with the pointers and lengths for your new items of data.

```
updatedAtomParam(outData->atomid,
                 (atmp_parameter*)ParamList->atmp_atomid);
updatedAtomParam(outData->published,
                 (atmp_parameter*)ParamList->atmp_published);
updatedAtomParam(outData->updated,
                 (atmp_parameter*)ParamList->atmp_updated);
updatedAtomParam(outData->edited,
                 (atmp_parameter*)ParamList->atmp_edited);
updatedAtomParam(outData->etagval,
                 (atmp_parameter*)ParamList->atmp_etagval);
updatedAtomParam(outData->nextsel,
                 (atmp_parameter*)ParamList->atmp_nextsel);
updatedAtomParam(outData->prevsel,
                 (atmp_parameter*)ParamList->atmp_prevsel);
updatedAtomParam(outData->firstsel,
                 (atmp_parameter*)ParamList->atmp_firstsel);
updatedAtomParam(outData->lastsel,
                 (atmp_parameter*)ParamList->atmp_lastsel);

if (strlen(outData->selector) != 0) {
    updatedAtomParam(outData->selector,
                     (atmp_parameter*)ParamList->atmp_selector);
}
```

- Uses the updateAtomContainer helper method to add your new data to the metadata and content containers, and set the option bit to indicate the presence of each container.

```
updateAtomContainer("DFHATOMTITLE    ",cChannel,AtomTitle,optPtr);
updateAtomContainer("DFHATOMSUMMARY  ",cChannel,AtomSummary,optPtr);
updateAtomContainer("DFHATOMCATEGORY ",cChannel,AtomCategory,optPtr);
updateAtomContainer("DFHATOMAUTHOR   ",cChannel,AtomAuthor,optPtr);
updateAtomContainer("DFHATOMAUTHORURI",cChannel,AtomAuthorUri,optPtr);
updateAtomContainer("DFHATOMEMAIL    ",cChannel,AtomEmail,optPtr);
updateAtomContainer("DFHATOMCONTENT  ",cChannel,AtomContent,optPtr);
```

- Provides a response code from the selection defined in the DFHW2CNH copybook. The reason code is ignored by CICS at present and reserved for future use, so its content is arbitrary.

```
ResponsePtr = (atmp_responses*)ParamList->atmp_response;
ResponsePtr->atmp_response_code = ATMP_RESP_NORMAL;
/*
 * ResponsePtr->atmp_response_code = ATMP_RESP_NOT_FOUND;
 * ResponsePtr->atmp_response_code = ATMP_RESP_NOT_AUTH;
 * ResponsePtr->atmp_response_code = ATMP_RESP_DISABLED;
 * ResponsePtr->atmp_response_code = ATMP_RESP_ALREADY_EXISTS;
 * ResponsePtr->atmp_response_code = ATMP_RESP_ACCESS_ERROR;
 */
ResponsePtr->atmp_reason_code = 0;
```

- Returns control to CICS automatically as the main program ends. CICS Atom processing uses the data from the metadata and content containers, and the data in the TWA for the DFHATOMPARMS parameters for which the service routine has supplied new pointers and lengths, to construct the Atom entry. CICS then requests further Atom entries from the service routine in the same way, until the window of entries to construct the Atom feed document is complete.

Chapter 23. Setting up CICS definitions for an Atom feed

To serve an Atom feed from CICS, set up URIMAP and ATOMSERVICE resource definitions, and an Atom configuration file. You can also set up an alternative alias transaction.

Before you begin

Before serving an Atom feed from CICS, you must configure the base components of CICS Web support to set CICS up as an HTTP server, if you have not already done this for your CICS region. Chapter 4, "Configuring CICS Web support base components," on page 53 explains how to do this.

Your CICS region must have a TCPIPSERVICE resource definition for the port where you want Web clients to make HTTP requests for your Atom feed. If you have already set up and used this CICS region as an HTTP server, it probably has a suitable TCPIPSERVICE resource, such as a definition for the well-known port number for HTTP, which is port 80. Define a new TCPIPSERVICE resource if you have not previously used this CICS region as an HTTP server and you have no TCPIPSERVICE resources, or if you want to use a nonstandard port number for your Atom feed that you have not yet defined in the CICS region. "Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 explains how to define a TCPIPSERVICE resource.

You must also choose the resource that provides the data for your Atom entries, and create either an XML binding or a service routine to support the delivery of this data. For more information about the options for resources, and instructions to create an XML binding or a service routine, see Chapter 22, "Setting up a resource to supply Atom entry data," on page 263.

About this task

Complete the tasks listed in this section to set up CICS definitions to serve an Atom feed that consists of data from your chosen resource, delivered using the XML binding or service routine that you have created.

Ensure that the configuration file is in EBCDIC and it is assumed that it is provided in codepage 1047 unless 'encoding=' has been specified.

When you have completed these tasks, you will have an Atom feed that Web clients can access to obtain a list of entries in the Atom format. The Web clients must process and display the data themselves. Many free or commercially available Web client applications are able to request, receive and display Atom feeds, including dedicated feed readers and also applications that provide further functions, such as applications for creating mashups. Check that the application is described as supporting the Atom format. You can also write your own Web client application to make GET requests for Atom feed data.

Related tasks:

"Making GET requests to Atom feeds or collections" on page 336
A Web client can obtain a list of the existing Atom entries in an Atom feed or collection by making an HTTP GET request to the URL of the collection, or obtain an individual Atom entry from the Atom feed or collection by making an HTTP GET request to the URL of the Atom entry.

Creating an alias transaction for an Atom feed

An alias transaction handles the later stages of processing for an Atom feed. CICS supplies a resource definition for a default Atom feed alias transaction, CW2A. Set up a TRANSACTION resource definition if you want to define an alternative alias transaction.

About this task

For non-Atom HTTP requests handled by CICS Web support, you only use an alias transaction when a user-written application program handles the requests. However, for Atom feeds, an alias transaction is used for processing all requests, whether or not a user-written service routine is involved.

You might want to use alternative alias transaction names for these purposes:

- Auditing, monitoring or accounting
- Modifying resource and command security settings
- Allocating initiation priorities
- Allocating DB2 resources
- Assigning different runaway values to different CICS application programs
- Transaction class limitation

You can set up as many alias transaction definitions as you want. You can use the URIMAP definition to specify the alias transaction that is required for a particular request.

CW2A specifies RESSEC(YES) and CMDSEC(YES), meaning that if resource and command security is active for the CICS region, it is applied to this transaction. If you specify resource and command security for your alias transaction, you will need to give Web clients appropriate permissions to access the resources and commands used by the transaction. For more information about security for Atom feeds and collections, see Chapter 26, "Security for Atom feeds," on page 357.

Follow the instructions in the *CICS Resource Definition Guide* to create a transaction resource definition. When you are following these instructions, note these points:

Procedure

- Base your alias transaction definition on the definition of CW2A, making any changes that you require. Here is the definition of CW2A:

```
DEFINE TRANSACTION(CW2A)   GROUP(DFHWEB2)
       PROGRAM(DFHW2A)     TWASIZE(512)
       PROFILE(DFHCICST)   STATUS(ENABLED)
       TASKDATALOC(ANY)    TASKDATAKEY(CICS)
       RUNAWAY(SYSTEM)     SHUTDOWN(DISABLED)
       PRIORITY(1)         TRANCLASS(DFHTCL00)
       DTIMOUT(NO)         TPURGE(NO)        SPURGE(YES)
       RESSEC(YES)         CMDSEC(YES)
       DESCRIPTION(CICS Web2.0 Atomservice alias transaction)
```

- Your alias transaction definition must use the CICS-supplied alias program DFHW2A. The alias program accesses the user-written service routine or CICS resource that is named in the ATOMSERVICE definition.
- Your alias transaction definition must be a local transaction.
- Make sure the priority of the alias transaction is equal to, or higher than, the priority of the transactions associated with Web attach tasks, such as CWXN or CWXU. The *CICS Performance Guide* explains why this is important.

Creating a URIMAP resource definition for an Atom document

Create a URIMAP resource to handle incoming requests from Web clients for the Atom document and specify the ATOMSERVICE resource for the document.

Before you begin

To support your URIMAP resource definition, you must have a TCPIPSERVICE definition that defines an inbound port for CICS Web support, on which CICS can receive HTTP requests. You can name a specific TCPIPSERVICE definition in your URIMAP resource definition, to specify a single port for the Atom feed requests. Alternatively, you can leave out the TCPIPSERVICE definition name to make the URIMAP resource definition apply for all inbound ports, but in this case your CICS region must have at least one suitable TCPIPSERVICE definition installed that defines an inbound port for HTTP requests. The TCPIPSERVICE resource definition is the place where you specify the security measures, such as SSL, that are applied for the port, and you can use these security measures to protect your Atom feeds and collections from unauthorized access. For instructions to set up a TCPIPSERVICE definition if your CICS region does not already have one that is suitable, see "Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96.

About this task

The CICS resource group DFH$WEB2 contains two sample URIMAP resource definitions for Atom collections, DFH$W2F1 and DFH$W2Q1. DFH$W2Q1 is used to deliver the sample Atom collection described in the Web 2.0 scenario "Create an Atom feed to work with employee information" in the CICS TS for z/OS, Version 4.1 Information Center, which is available at `https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp`.

The *CICS Resource Definition Guide* has information about the different methods of resource definition and full reference information about all the URIMAP resource definition attributes that you use during this process.

Procedure

1. Identify the URL that you plan to use for Web clients to obtain the Atom document. "URLs for Atom feeds from CICS" on page 245 explains how to construct a URL for an Atom feed. The URL must contain these items:
   - A scheme component of `http` or `https`.
   - A suitable host name for your CICS Web support implementation, such as `www.example.com`. If you specify a single asterisk as the HOST attribute, the URIMAP definition matches any host name on incoming URLs.
   - For an Atom feed document or collection, the beginning of a path component that can be extended to provide links to the whole document or to an individual entry, in Atom format or in an alternative format. You

specify this common part of the path component in the URIMAP resource definition with an asterisk as a wildcard (for example, `/myatomfeed/*`), and specify a complete path in the <atom:link> elements in the Atom configuration file for the feed. CICS uses the complete path to identify and return the appropriate item from the Atom feed.

Note: The common part of the path component must be unique to this Atom feed or collection among all the Atom feeds and collections that you serve using the host name that you specified. If you serve any other Web resources using the host name, the common part of the path component must not be the same as the beginning of the path for any of those resources.

- For an Atom service or category document, the complete path component for a suitable URL for the service or category document, such as `/servicedocument`.

  Note: The path for an Atom service or category document must not begin with the common part of the path component for any of the Atom feeds and collections that you serve using the host name that you specified.

2. Begin a URIMAP definition with a name and group of your choice, as specified in the *CICS Resource Definition Guide*.
3. Use the STATUS attribute to specify whether the URIMAP definition is installed in an enabled or disabled state.
4. Specify a USAGE attribute of ATOM.
5. Specify the SCHEME attribute as the scheme component of the URL for the HTTP request. You can use HTTP (without SSL) or HTTPS (with SSL). Do not include the delimiters `://` following the scheme component. A URIMAP specifying the HTTP scheme accepts Web client requests made using either the HTTP scheme or the more secure HTTPS scheme. A URIMAP specifying the HTTPS scheme accepts only Web client requests made using the HTTPS scheme.
6. If you need to distinguish between URLs containing different host names, specify the HOST attribute as the host component of the URL for the HTTP request. Do not include a port number. An IPv4 or IPv6 address can be used as a host name. If you specify a single asterisk as the HOST attribute, the URIMAP definition matches any host name on incoming URLs. Use this option if you are not using multiple host names or if you do not want to distinguish them.
7. For an Atom feed document or collection, specify the PATH attribute as the common part of the path component of the URL for the Atom feed requests, followed by an asterisk as a wildcard character. For an Atom service or category document, specify the PATH attribute as the complete path component for the document. You can either include or omit the delimiter `/` (forward slash) at the beginning of the path; if you omit it, CICS automatically provides it.
8. Optional: Specify the TCPIPSERVICE attribute as the name of the TCPIPSERVICE definition that defines the inbound port to which this URIMAP definition relates. If you do not specify this attribute, the URIMAP definition applies to a matching HTTP request on any inbound port. When a URIMAP definition with HTTPS (HTTP with SSL) 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.

9. Optional: If you set up an alternative alias transaction following the procedure in "Creating an alias transaction for an Atom feed" on page 294, specify the TRANSACTION attribute as the name of this alias transaction. The default alias transaction for Atom feeds is CW2A, in the DFHWEB2 group.
10. Specify the ATOMSERVICE attribute as a name for an ATOMSERVICE resource definition for the Atom document. The ATOMSERVICE resource identifies the resources that provide the data for the Atom document.
11. Optional: Specify the USERID attribute as a default user ID under which the alias transaction is attached. A user ID that you specify in the URIMAP definition is overridden by any authenticated user ID that is obtained from the Web client, using the authentication method specified by the AUTHENTICATE attribute of the TCPIPSERVICE definition for the connection. If no user ID is specified by either of these means, the default user ID is the CICS default user. CICS uses the user ID in security checking when resource and command security is active for the CICS region. For more information about security for Atom feeds, see Chapter 26, "Security for Atom feeds," on page 357. For an explanation of how the user IDs for Web clients are used in security checking, see "Resource and transaction security for application-generated responses" on page 180.

Creating an Atom configuration file for an Atom feed

Create your own Atom configuration file to provide metadata for the Atom feed and indicate the data that is provided by the CICS resource. You can base your Atom configuration file on the sample Atom configuration file `filea.xml`.

About this task

To create your Atom configuration file, copy the `filea.xml` sample Atom configuration file and change it to specify the information for your own Atom feed. When you install CICS Transaction Server, the sample Atom configuration file is installed in the `/samples/web2.0/atom` subdirectory of the root directory for CICS files on z/OS UNIX, as specified by the CICS system initialization parameter USSHOME. The default value for USSHOME is `/usr/lpp/cicsts/cicsts41`.

Rename your copy of the `filea.xml` sample, using the extension `.xml`, and store it in a z/OS UNIX System Services directory of your choice. You can edit your Atom configuration file using any XML editor or text editor.

In your Atom configuration file, make the following changes:

Procedure

1. In the root element <cics:atomservice>, change the attribute type="collection" to type="feed". The root element is specified in `filea.xml` as follows:

   ```
   <cics:atomservice type="collection"
        xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
        xmlns:atom="http://www.w3.org/2005/Atom">
   ```

   "<cics:atomservice> element" on page 301 has the reference information for this element.

2. In the <cics:feed> element, change the attribute window="6" if you want to change the number of Atom entries that CICS should return in each Atom feed document. The element as specified in `filea.xml` is as follows:

   ```
   <cics:feed window="6">
   ```

   If you remove the window attribute, CICS uses the default window of 8 entries. "<cics:feed> element" on page 302 has the reference information for this element.
3. In the <cics:resource> element, use the name and type attributes to specify the 1 - 16 character name and type of the CICS resource that provides the data for your Atom feed. `filea.xml` includes a <cics:resource> element that specifies the FILEA sample as follows:

   ```
   <cics:resource name="FILEA" type="file">
   </cics:resource>
   ```

   The resource name is specified in uppercase. The resource type can be "file", "tsqueue", or "program". If you are using a service routine to supply the data for your Atom feed, specify the name of the service routine and the resource type "program", not the name of the resource from which the service routine extracts the data. "<cics:resource> element" on page 302 has the reference information for this element.
4. In the <cics:bind> element, specify the name of the top-level data structure (the root element) of the XML binding that you set up for the resource that contains the data for your Atom entries.
   - If CICS is obtaining data directly from a file or temporary storage queue, specify the name of the root element from the XML binding for that file or temporary storage queue.
   - If you are using a service routine to supply the data for your Atom feed, and you were able to set up an XML binding for the resource from which the service routine extracts the data, specify the name of the root element from that XML binding.
   - If you are using a service routine and you do not have an XML binding for the resource that holds the data, omit the <cics:bind> element.

   `filea.xml` specifies the XML binding for the FILEA sample as follows:

   ```
   <cics:bind root="DFH0CFIL"/>
   ```

   "<cics:resource> element" on page 302 has the reference information for the <cics:bind> element.
5. In the <cics:authority> element, specify a suitable authority name and date that CICS can use to generate unique Atom IDs for your Atom feed and Atom entries. `filea.xml` includes an example of the <cics:authority> element as follows:

   ```
   <cics:authority name="example.com" date="2009-02-14"/>
   ```

   "<cics:authority> element" on page 303 explains how to choose an authority name and date, and what to do if you want to use an alternative format for your Atom IDs.
6. Change the <cics:selector> element if you require compatibility with an application developed using the CA8K SupportPac, or if the selector value for your Atom entries cannot be represented as a character string. The selector style specified in the sample Atom configuration file is the default "segment" style, which uses a standard URL style for individual Atom entries:

   ```
   <cics:selector style="segment"/>
   ```

"<cics:selector> element" on page 304 has the reference information for this element.

7. If the resource that contains the data for your Atom entries has fields in its records to hold metadata for the Atom entries, such as titles or time stamps, add the <cics:fieldnames> element as a child element of the <cics:resource> element, and use its attributes to specify the names of the fields that contain the metadata. "<cics:fieldnames> element" on page 306 has information about this element. The sample Atom configuration file does not include this element because the FILEA file, which provides the data for the sample Atom feed, does not contain any fields that hold metadata for Atom entries. If the resource that contains your Atom entries also does not have any metadata, do not add the <cics:fieldnames> element.
8. In the <atom:link rel="self" href=" "> element that is a child element of the <atom:feed> element, specify the URL that Web clients can use to retrieve the Atom feed document. The beginning of the path must match the partial path that you stated in the URIMAP resource definition for the feed. For example, if you specified `/myatomfeed/*` as the path component in the URIMAP resource definition, you could specify <atom:link rel="self" href="/myatomfeed/feed"> in the Atom configuration file. `filea.xml` shows the following example link:

   ```
   <atom:link rel="self" href="/atom/f/filea/feed"/>
   ```

   "<atom:feed> element" on page 308 has the reference information for all the child elements of the <atom:feed> element.
9. Change the other child elements of the <atom:feed> element to specify the metadata for your Atom feed, such as the title and the name of the primary author. `filea.xml` includes an appropriate selection of child elements as follows:

   ```
   <atom:title>Sample CICS file FILEA</atom:title>
   <atom:subtitle>A RESTFUL service for FILEA</atom:subtitle>
   <atom:icon>../../../web2.0/image/cics-icon.gif</atom:icon>
   <atom:logo>../../../web2.0/image/cics-logo.gif</atom:logo>
   <atom:rights>Copyright (c) Example Corp 2009</atom:rights>
   <atom:category term="Demonstration"/>
   <atom:author>
       <atom:name>CICS Development</atom:name>
       <atom:email>cics@example.com</atom:email>
       <atom:uri>http://www.example.com/cics</atom:uri>
   </atom:author>
   ```

   "<atom:feed> element" on page 308 lists all the possible child elements, and states whether or not each element is required by the Atom specifications.
10. In the <atom:link rel="self" href=" "> element that is a child element of the <atom:entry> element, specify the standard path that CICS can use to create URLs for individual Atom entries. CICS creates a URL by appending the selector value for an Atom entry to this standard path. The beginning of the path must match the partial path that you stated in the URIMAP resource definition for the feed. The remainder of the standard path must be different from the path for the feed. For example, if you specified `/myatomfeed/*` as the path component in the URIMAP resource definition, and <atom:link rel="self" href="/myatomfeed/feed.atom"> as the link for the whole feed in the Atom configuration file, you could specify <atom:link rel="self" href="/myatomfeed/entries/"> as the standard path for entries. For an Atom entry with a selector value of 23, CICS would create the link <atom:link rel="self" href="/myatomfeed/entries/23">, if you specified the segment style in the <cics:selector> element. `filea.xml` shows the following example link:

```
<atom:link rel="self" href="/atom/f/filea"/>
```

"<atom:entry> element" on page 311 has the reference information for all the child elements of the <atom:entry> element.

11. Change the metadata child elements of the <atom:entry> element to specify suitable default metadata for your Atom entries, such as a default title. `filea.xml` includes an appropriate selection of child elements as follows:

```
<atom:title>FILEA item</atom:title>
<atom:author>
   <atom:name>CICS Development</atom:name>
   <atom:uri>http://www.example.com/cics</atom:uri>
</atom:author>
<atom:rights>Copyright (c) Example Corp 2009</atom:rights>
<atom:published>2009-02-28T12:00:00Z</atom:published>
```

"<atom:entry> element" on page 311 lists all the possible child elements, and states whether or not each element is required by the Atom specifications.

12. In the <atom:content> child element of the <atom:entry> element:
    a. Use the cics:resource and cics:type attributes to specify the 1 - 16 character name and type of the CICS resource or service routine that provides the content for the Atom entries in the feed, exactly as you specified them in the <cics:resource> element. `filea.xml` includes an <atom:content> element that specifies the FILEA sample as follows:

```
<atom:content cics:resource="FILEA" cics:type="file"/>
```

    b. Add the type attribute and use it to specify the media type for the content of the Atom entries in the feed. Some possible type attributes are as follows:
       - type="text" specifies plain text.
       - type="html" specifies HTML.
       - type="xhtml" specifies XHTML.
       - type="text/xml" is the media type that is normally used for a human-readable XML document.

       If you omit this attribute, CICS uses a default media type of "application/xml". CICS labels the content with the media type or the default when the Atom document is issued. If you are using a service routine, you can override it and specify an alternative media type.

"<atom:entry> element" on page 311 has the reference information for the <atom:content> child element.

13.

Example

The contents of `filea.xml` are shown here without the comments that are in the sample, to demonstrate the nesting of the XML elements:

```
<?xml version="1.0"?>
<cics:atomservice type="collection"
     xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
     xmlns:atom="http://www.w3.org/2005/Atom">
   <cics:feed window="6">
     <cics:resource name="FILEA" type="file">
         <cics:bind root="DFH0CFIL"/>
     </cics:resource>
     <cics:authority name="example.com" date="2009-02-14"/>
     <cics:selector style="segment"/>
   </cics:feed>
```

```
   <atom:feed>
     <atom:link rel="self" href="/atom/f/filea/feed"/>
     <atom:title>Sample CICS file FILEA</atom:title>
     <atom:subtitle>A RESTFUL service for FILEA</atom:subtitle>
     <atom:icon>../../../web2.0/image/cics-icon.gif</atom:icon>
     <atom:logo>../../../web2.0/image/cics-logo.gif</atom:logo>
     <atom:rights>Copyright (c) Example Corp 2009</atom:rights>
     <atom:category term="Demonstration"/>
     <atom:author>
         <atom:name>CICS Development</atom:name>
         <atom:email>cics@example.com</atom:email>
         <atom:uri>http://www.example.com/cics</atom:uri>
     </atom:author>
     <atom:entry>
        <atom:link rel="self" href="/atom/f/filea"/>
        <atom:title>FILEA item</atom:title>
        <atom:author>
           <atom:name>CICS Development</atom:name>
           <atom:uri>http://www.example.com/cics</atom:uri>
        </atom:author>
        <atom:rights>Copyright (c) Example Corp 2009</atom:rights>
        <atom:published>2009-02-28T12:00:00Z</atom:published>
        <atom:content cics:resource="FILEA" cics:type="file"/>
     </atom:entry>
   </atom:feed>
</cics:atomservice>
```

What to do next

When you have set up your Atom configuration file, create an ATOMSERVICE resource definition that references it, following the instructions in "Creating an ATOMSERVICE definition for an Atom feed" on page 318.

<cics:atomservice> element

The <cics:atomservice> element is the root element for an Atom configuration file.

Attributes

type=*typevalue*
The type of Atom document that is being configured. This attribute is required. *typevalue* must match the document type specified in the ATOMTYPE attribute of the ATOMSERVICE resource definition that references this Atom configuration file.

type="feed"
An Atom feed document.

type="service"
An Atom service document.

type="collection"
An Atom collection document.

type="category"
An Atom category document.

xmnls:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
The namespace declaration to bind the configuration file to the CICS Atom XML namespace. Do not change this namespace declaration.

xmnls:atom="http://www.w3.org/2005/Atom"
The namespace declaration to bind the configuration file to the Atom XML namespace. Do not change this namespace declaration.

The namespace declaration **xmnls:app**="**http://www.w3.org/2007/app**" is used to bind an Atom document to the namespace for the Atom Publishing Protocol. However, this namespace declaration does not appear in an Atom configuration file for an Atom feed or collection, because those Atom configuration files do not contain any elements with the app: prefix. When CICS uses the <app:edited> element in an Atom document that it serves for a collection, CICS adds this namespace declaration automatically. The namespace declaration is required in an Atom configuration file for an Atom service or category document.

Contains:

"<cics:feed> element"

"<atom:feed> element" on page 308

Example

```
<cics:atomservice type="feed"
      xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
      xmlns:atom="http://www.w3.org/2005/Atom">
</cics:atomservice>
```

<cics:feed> element

The <cics:feed> element in an Atom configuration file contains other elements that describe the CICS resource that is to be published as a feed. It also specifies the window for the number of entries that CICS is to return in each feed document, and contains elements to specify the style of the request URLs and Atom IDs for entries.

Contained by:

"<cics:atomservice> element" on page 301

Attributes

window="*number***"|"8"**

The default number of entries that CICS is to return in the Atom feed document. This attribute is optional, and the default window is 8 entries. A client can specify a different window size. The window size only applies when the client makes a request using the URL for the whole Atom feed, or a navigation URL for a partial list of Atom entries. When a client makes a request using the URL of an individual Atom entry, CICS returns only the single requested Atom entry.

Contains:

"<cics:selector> element" on page 304

"<cics:authority> element" on page 303

"<cics:resource> element"

Example

```
<cics:feed window="10">
</cics:feed>
```

<cics:resource> element

The <cics:resource> element in an Atom configuration file specifies the name and type of the CICS resource that is to be published as a feed.

Contained by:

"<cics:feed> element" on page 302

Attributes

name="*cics-resource-name***"**

The name of the CICS resource that provides data for the feed. This attribute is required. The name is case-sensitive, so uppercase is normally correct. The resource name must match the value of the RESOURCENAME attribute of the ATOMSERVICE resource definition that references this Atom configuration file.

type="*cics-resource-type***"**

The type of the CICS resource. This attribute is required. The resource type is specified in lowercase. The resource type must match the value of the RESOURCETYPE attribute of the ATOMSERVICE resource definition that references this Atom configuration file.

type="tsqueue"
A temporary storage queue.

type="file"
A file.

type="program"
A program (service routine).

Contains:

<cics:bind> element

The <cics:bind> element has one attribute **root**="*root-element-name*", which specifies the name of the top-level data structure in the XML binding.

"<cics:fieldnames> element" on page 306

Example

```
<cics:resource name="feedq" type="tsqueue">
     <cics:bind root="SAMPBIND"/>
</cics:resource>
```

<cics:authority> element

The <cics:authority> element in an Atom configuration file provides the authority name and associated date that CICS uses when creating tag URIs to use as Atom IDs for individual Atom entries.

The other elements of the tag URI are a prefix of "tag", and a specific consisting of the resource type and resource name that you specify in the <cics:resource> element in the Atom configuration file, and the selector value for the individual Atom entry. "Atom IDs for Atom entries" on page 255 explains the tag URI format and the requirements for the use of Atom IDs.

If you prefer to use an alternative format for the Atom IDs, use the <atom:id> element in the prototype Atom entry to specify a prototype Atom ID, and omit the <cics:authority> element. If you use an alternative Atom ID format, make sure that the Atom IDs are unique and meet the requirements of the Atom format specification in RFC 4287.

The tag URIs are unique within a CICS region, but they are not guaranteed to be unique across different CICS regions. In the situation where you want to set up Atom feeds from resources that have the same name and type but reside on

different CICS regions, specify a different authority name or a different date in the <cics:authority> element of the Atom configuration file for each of the feeds. Tag URIs that have different dates are not equivalent to each other, even if all the other information is the same.

The tag URIs are unique for Atom entries provided by a user-written service routine that deals with a single Atom feed, but they are not unique if the user-written service routine provides more than one feed, because the name of the service routine is used as the resource name in the tag URIs. If your user-written service routine provides multiple feeds, either choose an alternative format for your Atom IDs, or use a different authority name or a different date in the <cics:authority> element of the Atom configuration file for each of the feeds.

Contained by:

"<cics:feed> element" on page 302

Attributes

name="*authority-name***"**
The authority name. To comply with RFC 4151, the authority name must be a fully qualified domain name or email address that is registered to you or to your company. You must not use a domain name or email address owned by someone else. RFC 4151 recommends that you specify the authority name in lower case, and tag URIs where the same authority name is specified in a different case are not equivalent to each other. You should ensure that you have your company's agreement to use a domain name that others in the company might also use to generate tag URIs.

date="*YYYY-MM-DD***"**
A date on which the authority name was owned by you or your company. You can use any date between the time when you first owned the authority name and the present day, but RFC 4151 recommends that you do not use the first day of ownership or a few days immediately following it. You must not use a future date. CICS checks that your date is a valid past or present date.

Example

```
<cics:authority name="example.com" date="2009-01-08"/>
```

<cics:selector> element

The <cics:selector> element in an Atom configuration file identifies the nature and position within the URL of the selector value for an Atom entry. CICS uses this information to locate and extract the selector value in an HTTP request for a specific Atom entry, and to generate a URL in the correct format when issuing an Atom entry as part of an Atom feed document or collection.

Contained by:

"<cics:feed> element" on page 302

Attributes

style="*style-type***"**
The styles are as follows:

"segment"
<cics:selector style="segment"/> represents a standard URL style for

individual Atom entries, where the selector value for the Atom entry is placed as the final segment of the path component of the URL. In this URL, the selector value is "25":

```
http://www.example.com/web20/sample_atom_feed/entry/25
```

"segment" is the default style, so if you require this style and your selector value is in the format that CICS assumes for the resource type, you may omit the <cics:selector> element from your Atom configuration file.

"query"
<cics:selector style="query"/> represents a URL style that is compatible with applications developed using the CA8K SupportPac, where the selector value for the Atom entry is placed in a query string. In this URL, the selector value is "25":

```
http://www.example.com/web20/sample_atom_feed/entry?s=25
```

name="*query-parm-name***"**
If you select style="query", you may use this attribute to specify the name of the query string keyword that identifies the selector value. If you omit the name attribute, the default keyword "s" is used. Do not use this attribute when you select style="segment".

format="*format-name***"**
This attribute identifies the format of the selector value. If your selector value is in the format that CICS assumes for the type of resource that contains your Atom entries, do not specify this attribute. Specify the attribute only if your resource contains a nonstandard selector value. The formats are as follows:

"decimal"
The selector value is a decimal number. CICS assumes this format is used for temporary storage queues and for RRDS and VRRDS files. Specify this setting if you are using any other type of resource and it has a decimal number as the selector value.

"hexadecimal"
The selector value is a binary number. CICS assumes this format is used for ESDS and extended ESDS files. Specify this setting if you are using any other type of resource and it has a binary number as the selector value.

For any other type of VSAM file, CICS assumes that the format of the selector value is a character string. You cannot specify character string format explicitly using the attribute.

ccsid="*nnnn***"**
This attribute specifies the coded character set identifier (CCSID, or code page) into which the selector value must be converted before being passed to the service routine. Do not specify this attribute if format="decimal" or format="hexadecimal" is specified, or if CICS assumes one of those formats for your resource. Specify the attribute only if your selector value is in character string format, and the selector value might contain non-alphanumeric characters that are percent-encoded in the request URL. "*nnnn*" is the number of an EBCDIC CCSID into which percent-encoded UTF-8 characters are to be converted. If CICS encounters percent-encoded characters in the selector value but the ccsid attribute is omitted, CICS uses the value specified in the LOCALCCSID system initialization parameter. The CCSIDs that CICS supports are referenced in the description of the LOCALCCSID system initialization parameter in the *CICS System Definition Guide*.

Example

```
<cics:selector style="query" name="sel" format="hexadecimal"/>
```

<cics:fieldnames> element

The <cics:fieldnames> element in an Atom configuration file identifies the field names in records in the CICS resource that provide items of metadata or content for the Atom entry documents. The attributes of the element specify the field names that have significance.

Contained by:

"<cics:resource> element" on page 302

Attributes

To support the use of the <cics:fieldnames> element, you must have an XML binding for the resource that contains the data for your Atom entries. The *CICS Application Programming Guide* explains how to create this. In the <cics:fieldnames> element, specify the names of the fields using the XML names that are produced by the CICS XML assistant when you create the XML binding, not the original field names as stated in the high-level language structure, or copybook, that describes the structure of the resource.

For CICS resource types of files or temporary storage queues where CICS extracts the data for the Atom entries directly from the resource, you must use the <cics:fieldnames> element if the records in your file or temporary storage queue include any fields that hold metadata for the Atom entries, such as titles or time stamps. If your records contain any metadata, add the <cics:fieldnames> element, and use the attributes of the element to specify the fields in the records in your CICS resource that provide metadata for the Atom entries. If the records in your CICS resource contain only the content of the entries and do not have any metadata, omit the <cics:fieldnames> element.

If you are using a program, known as a service routine, to provide the Atom entries, add the <cics:fieldnames> element if you are using the resource handling parameters in the DFHATOMPARMS container to pass the field names to the program. Use the attributes of the element to name the fields in the records in the resource that the program accesses to obtain metadata and content for the Atom entries. You must have an XML binding for the resource in order to do this. If your program holds its own information about the resource structure and does not use the resource handling parameters, omit the <cics:fieldnames> element.

All of the attributes of the <cics:fieldnames> element are optional. If you do not specify a particular attribute, such as the author's name, CICS either supplies that item of metadata from the corresponding element in the <atom:entry> element in the configuration file or omits it. If you omit all the attributes, you can also omit the <cics:fieldnames> element.

atomid="*fieldname***"**
The name of a field in the resource record that contains the unique identifier for the Atom entry.

author="*fieldname***"**
The name of a field in the resource record that contains the personal name of the principal author of the Atom entry. You cannot provide the details of additional authors or contributors from fields in the CICS resource record. You can provide these in the <atom:feed> element in the configuration file, but they

apply to all the entries. If you provide author details from the CICS resource using the <cics:fieldnames> element, and also specify one or more instances of <atom:author> in the configuration file, the author details from the CICS resource are placed before the author details from the configuration file.

authoruri="*fieldname***"**
The name of a field in the resource record that contains a URL associated with the principal author of the Atom entry, such as a blog site or a company Web site.

category="*fieldname***"**
The name of a field in the resource record that contains a category that classifies the entry.

content="*fieldname***"**
The name of a field in the resource record that contains the entire content to be published in the Atom entry. The name can be that of a substructure within the record, in which case the whole substructure is propagated into the published content as an XML element with child elements, if necessary. If you omit this attribute, CICS publishes the whole of the resource record as the content of the entry.

content_type="*fieldname***"**
The name of a field in the resource record that contains the media type for the content of the Atom entry.

edited="*fieldname***"**
The name of a field in the resource record that contains the timestamp that indicates when the record was last edited. The data type of the named field is obtained from the description in the XML binding specified by the BINDFILE attribute of the ATOMSERVICE resource definition. If the named field is a character string of length at least 20, CICS assumes that it contains a timestamp in the XML dateTime format, as described in RFC 3339; for example, "2008-05-20T11:39:50.325Z". If the named field is a packed decimal field of length 8, and the optional DATETIME=PACKED15 parameter was used when the XML binding was prepared, CICS assumes that the field contains a CICS ABSTIME value.

email="*fieldname***"**
The name of a field in the resource record that contains the e-mail address of the principal author of the Atom entry.

published="*fieldname***"**
The name of a field in the resource record that contains the timestamp or ABSTIME value that that indicates when the record was first published. CICS identifies the format of the data in the same way as for the edited attribute.

title="*fieldname***"**
The name of a field in the resource record that contains the title for the Atom entry.

summary="*fieldname***"**
The name of a field in the resource record that contains the summary for the Atom entry.

updated="*fieldname***"**
The name of a field in the resource record that contains the timestamp or ABSTIME value that indicates when the record was last updated. CICS identifies the format of the data in the same way as for the edited attribute.

Example

These field names are the XML versions of the field names in the example COBOL language structure shown in “Creating a CICS resource to store Atom entries” on page 264.

```
<cics:fieldnames title="title_field"
                 summary="summary_field"
                 atomid="atomid_field"
                 content="content_field"
                 author="author_name_field"
                 email="author_email_field"
                 authoruri="author_uri_field"
                 edited="edited_field"
                 updated="updated_field"
                 published="published_field"
                 category="category_field"/>
```

Related reference:

“DFHATOMPARMS container” on page 271
DFHATOMPARMS is a container of DATATYPE(CHAR) that contains parameters that CICS uses to communicate with a service routine that provides data for an Atom feed.

<atom:feed> element

The <atom:feed> element in an Atom configuration file is a prototype for the Atom feed document that CICS returns. It provides the metadata for the Atom feed, and contains a single prototype <atom:entry> element.

Contained by:

“<cics:atomservice> element” on page 301

Child elements

The child elements of the <atom:feed> element are as defined by the Atom format specification in RFC 4287. Some of the child elements are required and some of them are optional.

RFC 4287 permits plain text, HTML, or XHTML content for child elements that are defined as "text constructs", such as the <atom:title> element. CICS only supports plain text content for these elements, so you cannot use a "type" attribute to specify an alternative content type. You must supply plain text content with no child elements. CICS does permit HTML, XHTML, and other text media types (such as XML) in the <atom:content> element as content for Atom entries, which you specify in the CICS resource that provides data for the Atom entries.

The child elements for the <atom:feed> element are as follows:

<atom:author>
The personal details of the principal author of the Atom feed, which might be an individual person or an organization. You can have more than one <atom:author> element in the configuration file. The data is provided in child elements as follows:

<atom:name>
The name of the person. This child element is required if you are specifying the <atom:author> element, and CICS checks that you include it.

<atom:uri>
An URL associated with the person, such as a blog site or a company web site. This child element is optional. CICS checks that you do not specify more than one <atom:uri> element in an <atom:author> element. CICS does not attempt to verify that the URL is valid, so you must ensure that it is correct.

<atom:email>
The e-mail address of the person. This child element is optional. CICS checks that you do not specify more than one <atom:email> element in an <atom:author> element.

The <atom:author> element is optional. If you choose not to specify the <atom:author> element anywhere in your configuration file, you must ensure that all the Atom entries in your resource include this data, in order to be compliant with RFC 4287.

<atom:category term=" ">
The name of a category that classifies the Atom feed. This element is optional. CICS only supports a single instance of this element. The term attribute specifies the name of the category.

<atom:contributor>
The personal details of a subsidiary author of the Atom feed. This element is optional. You can have more than one of this element in the configuration file. The data is provided in child elements as follows:

<atom:name>
The name of the person. This child element is required when you use the <atom:contributor> element.

<atom:uri>
An URL associated with the person, such as a blog site or a company Web site. This child element is optional.

<atom:email>
The e-mail address of the person. This child element is optional.

<atom:entry>
In an Atom configuration file, you need a single prototype <atom:entry> element. The description of this element is in "<atom:entry> element" on page 311.

<atom:generator>
The name of the agent that generates the Atom feed. Do not specify this element in the Atom configuration file; CICS provides it when composing the Atom feed document. CICS provides an identification of itself as the generator of the Atom feed.

<atom:icon>
An URL that points to a small icon representing the Atom feed. This element is optional. CICS checks that you do not specify more than one <atom:icon> element for the Atom feed, but does not check the aspect ratio of the image, which should be 1 (horizontal) to 1 (vertical).

<atom:id>
The unique identifier for the Atom feed. The Atom format specification requires one <atom:id> element for the Atom feed. If you specified the <cics:authority> element in the Atom configuration file to make CICS generate tag URIs as Atom IDs, you can omit the <atom:id> element for the Atom feed, and CICS generates an Atom ID for the Atom feed in the

same format as for the Atom entries, but without the selector value or unique identifier that is appended for the Atom entries. For example:

```
tag:example.com,2009-01-08:tsqueue:WB20TSQ
```

If you prefer an alternative Atom ID format, or if you are using the <atom:id> element in the prototype Atom entry to specify an Atom ID format and you want to copy this, include the <atom:id> element for the Atom feed and specify a complete Atom ID for the Atom feed. Make sure that the Atom ID is unique and meets the requirements of the Atom format specification in RFC 4287.

<atom:link>
A URL that identifies the Atom feed document and enables Web clients to retrieve it. "URLs for Atom feeds from CICS" on page 245 explains how to construct this URL.

An Atom feed document must have a single <atom:link rel="self"> element as a child element of the <atom:feed> element. The href attribute contains a URL that Web clients can use to retrieve the Atom feed document. CICS does not provide support for other types of <atom:link> element for Atom feed documents.

In your Atom configuration file, the <atom:link rel="self"> element must state the complete path that Web clients can use to retrieve the Atom feed document, with the beginning of the path matching the partial path that you stated in the URIMAP resource definition for the Atom feed or collection. For example, if you specified `/myatomfeed/*` as the path component in the URIMAP resource definition, you could specify <atom:link rel="self" href="/myatomfeed/feed.atom"> in the Atom configuration file. The limits on URL length listed in "URLs for CICS Web support" on page 33 apply also to URLs for Atom feeds.

In the Atom configuration file, you may omit the scheme and host components of the URL, and specify only the path component. CICS adds the scheme and host components to the URL when it returns the Atom feed or Atom entry document to the client, to comply with the Atom format specification.

<atom:logo>
An URL that points to a larger logo representing the Atom feed. This element is optional. CICS checks that you do not specify more than one <atom:logo> element for the Atom feed, but does not check the aspect ratio of the image, which should be 2 (horizontal) to 1 (vertical).

<atom:rights>
A text string that contains the claimed intellectual property rights, such as copyright. CICS only supports plain text for this element. This element is optional.

<atom:subtitle>
The subtitle for the Atom feed. CICS only supports plain text for subtitles. This element is optional. CICS checks that you do not specify more than one <atom:subtitle> element for the Atom feed.

<atom:title>
The title for the Atom feed. CICS only supports plain text for titles. You must specify a <atom:title> element to comply with RFC 4287. CICS checks that you do not specify more than one <atom:title> element.

<atom:updated>

The time at which the Atom feed was last updated. Do not specify this element in the Atom configuration file; CICS provides it when composing the Atom feed document. CICS supplies this timestamp as the most recent of all the <atom:updated> elements of the enclosed <atom:entry> elements that make up the Atom feed. If the CICS resource containing the Atom entries does not provide this data, CICS defaults to the current date and time.

Contains:

"<atom:entry> element"

Example

```
<atom:feed>
    <atom:title>CICS Atom feed</atom:title>
    <atom:subtitle>My first Atom feed from CICS</atom:subtitle>
    <atom:link rel="self" href="/web20/sample_atom_feed" />
    <atom:rights>Copyright (c) 2009, Joe Bloggs</atom:rights>
    <atom:author>
         <atom:name>Joe Bloggs</atom:name>
         <atom:uri>http://www.ibm.com/JBloggs/</atom:uri>
         <atom:email>JBloggs@uk.ibm.com</atom:email>
     </atom:author>
     <atom:contributor>
         <atom:name>John Doe</atom:name>
     </atom:contributor>
</atom:feed>
```

<atom:entry> element

The <atom:feed> element in an Atom configuration file contains a single prototype <atom:entry> element. CICS generates an Atom entry document by populating the child elements of this element with data supplied by the CICS resource described in the <cics:resource> element, repeating this process for as many Atom entries as the client requests.

Contained by:

"<atom:feed> element" on page 308

Child elements

The child elements of the <atom:entry> element are as defined by the Atom format specification in RFC 4287, except that CICS does not support the optional <atom:source> element. Some of the child elements are required for the Atom entry, and some of them are optional.

Most of the child elements of the <atom:entry> element contain default metadata to provide any data items for an individual Atom entry that are not supplied by the CICS resource or service routine.

- If your CICS resource or service routine always supplies data for one of these child elements, you can typically omit it from the <atom:entry> element in your configuration file. The exception is the <atom:title> element, which CICS requires, although you may use an empty element if your Atom entries all have a title and no default would be suitable.
- If your Atom entries are held in a resource whose records are lacking fields for some or all of the possible items of metadata, you can supply these items of metadata by specifying them in the Atom configuration file. When you provide

metadata in the <atom:entry> element, CICS uses this metadata whenever the corresponding item of data is not supplied by the CICS resource or service routine.

- You can omit any items of metadata that are optional, if you do not want to provide a default for these items.

Some of the child elements of the <atom:entry> element correspond to the attributes of the <cics:fieldnames> element. If a child element has a corresponding attribute in the <cics:fieldnames> element, CICS can deliver that item of data from the resource, or a service routine can deliver it, if it is present. If a child element has no corresponding attribute in the <cics:fieldnames> element, you can only specify it in the Atom configuration file.

RFC 4287 permits plain text, HTML, or XHTML content for child elements that are defined as "text constructs", such as the <atom:title> element. CICS only supports plain text content for these elements, so you cannot use a "type" attribute to specify an alternative content type. You must supply plain text content with no child elements. CICS does permit HTML, XHTML, and other text media types (such as XML) in the <atom:content> element as content for Atom entries, which you specify in the CICS resource that provides data for the Atom entries.

The child elements for the <atom:entry> element are as follows:

<app:edited>
The time when the Atom entry was last edited. This element applies only to Atom entries that are part of a collection, and in that case it is required (as a SHOULD requirement). You cannot specify the <app:edited> element in your prototype Atom entry in the configuration file. The element corresponds to the edited attribute of the <cics:fieldnames> element. If the CICS resource or service routine does not provide this data, CICS supplies the current date and time as a default. Although CICS supports the use of the <app:edited> element, for reasons of performance CICS does not automatically order Atom entries in a collection by this element when returning them to a client as a list. For more information about ordering Atom entries in a collection, see “Sequence for Atom entries” on page 252.

<atom:author>
The personal details of the principal author of the Atom element, which might be an individual person or an organization. You can have more than one <atom:author> element in the configuration file. The data is provided in child elements as follows:

<atom:name>
The name of the person. This child element is required if you are specifying the <atom:author> element, and CICS checks that you include it.

<atom:uri>
An URL associated with the person, such as a blog site or a company web site. This child element is optional. CICS checks that you do not specify more than one <atom:uri> element in an <atom:author> element. CICS does not attempt to verify that the URL is valid, so you must ensure that it is correct.

<atom:email>
The e-mail address of the person. This child element is optional. CICS checks that you do not specify more than one <atom:email> element in an <atom:author> element.

The <atom:author> element is optional. If you choose not to specify the <atom:author> element anywhere in your configuration file, you must ensure that all the Atom entries in your resource include this data, in order to be compliant with RFC 4287.

<atom:category term=" ">
The name of a category that classifies the Atom entry. This element is optional. CICS only supports a single instance of this element. The term attribute specifies the name of the category. The element corresponds to the category attribute of the <cics:fieldnames> element, so you can omit the element if your CICS resource always provides suitable data.

<atom:content type=" " cics:resource=" " cics:type=" "/>
The content of the Atom entry. This element is required.

The <atom:content> element as specified in the configuration file does not contain any data, because CICS supplies the data from the resource when it issues the feed document. You must ensure that your resource provides content for every Atom entry. CICS does not support the delivery of Atom entries that contain no data, such as Atom entries that use the "src" attribute to reference remote content.

type=" "
The type attribute specifies the type of content that CICS expects for the Atom entry, which can be "text", "html", "xhtml", or an IANA media type. If this attribute is not present, CICS uses a default media type of "application/xml". As in RFC 4287, use the media type "text" for plain text instead of the IANA media type "text/plain", "html" instead of "text/html", and "xhtml" instead of "application/xhtml+xml". If your content is in any other format, specify the IANA media type that you would normally use for that format on the Internet. A listing of media types is available at http://www.iana.org/assignments/media-types/. Note that CICS does not provide support for nontext media types.

When CICS delivers Atom entries directly from your resource, the media type or the default must be appropriate for the data in the resource, because CICS labels the content with this media type when the Atom document is issued. When you use a service routine to provide content for the Atom entries, this media type or the default is supplied to the service routine. The service routine can override it and specify an alternative media type, or allow CICS to label the content with the media type from the Atom configuration file.

" cics:resource=" and " cics:type=" "
The attributes cics:resource and cics:type state the name and type of the CICS resource that provides the data for the Atom feed. The values of these attributes must match the values of the name and type attributes stated for the <cics:resource> element. For a description of the values of these attributes, see “<cics:resource> element” on page 302.

The <atom:content> element corresponds to the content and content_type attributes of the <cics:fieldnames> element.

<atom:contributor>
The personal details of a subsidiary author of the Atom entry. This element is optional. The data is provided in the <atom:name>, <atom:uri>, and <atom:email> child elements as in the <atom:feed> element. You cannot

specify data for <atom:contributor> elements in individual resources, so the <cics:fieldnames> element has no corresponding attribute, and the data that you provide applies to all Atom entries. Because the data applies to all Atom entries, you might prefer to specify contributors under the <atom:feed> element.

<atom:id>
The unique identifier for the Atom entry. The Atom format specification requires one <atom:id> element for each Atom entry. “Atom IDs for Atom entries” on page 255 explains the requirements for the use of Atom IDs.

CICS can generate tag URIs to use as unique Atom IDs for Atom entries. The tag URIs include the attributes that you specify in the <cics:authority> element in the Atom configuration file, the resource type and resource name that you specify in the <cics:resource> element, and the selector value for the individual Atom entry. If this format meets your needs, omit the <atom:id> element in the prototype Atom entry.

Instead of using the tag URI format that is generated by CICS, you may use the <atom:id> element in the prototype Atom entry to specify a prototype Atom ID in an alternative format. CICS appends the selector value or a suitable unique identifier to produce a unique Atom ID. If you use an alternative Atom ID format, make sure that the Atom IDs are unique and meet the requirements of the Atom format specification in RFC 4287.

<atom:link>
A standard URL that identifies an Atom entry and enables Web clients to retrieve it. “URLs for Atom feeds from CICS” on page 245 explains how to construct this URL.

For CICS, you must have a single <atom:link rel="self"> element as a child element of the <atom:entry> element. For an Atom entry in a collection, the Atom format specification requires a link relation of "edit" instead of "self", and CICS supplies this link relation automatically when sending out Atom entries in a collection. You must specify <atom:link rel="self"> in the Atom configuration file, whether the Atom entries are in an Atom feed or a collection.

In your Atom configuration file, specify the URL in an <atom:link rel="self"> element as a standard path that can be extended to apply to any Atom entry document. The beginning of the path must match the partial path that you stated in the URIMAP resource definition for the Atom feed or collection. The remainder of the standard path must be different from the complete path that you specified in the <atom:link rel="self"> element for the Atom feed. For example, if you specified `/myatomfeed/*` as the path component in the URIMAP resource definition, and <atom:link rel="self" href="/myatomfeed/feed.atom"> as the link for the whole Atom feed in the Atom configuration file, you could specify <atom:link rel="self" href="/myatomfeed/entries/"> as the standard path for Atom entries. The limits on URL length listed in “URLs for CICS Web support” on page 33 apply also to URLs for Atom feeds.

When CICS returns an Atom entry document to the client, CICS appends the selector value for the Atom entry to this path to create a complete link.“Selector value for Atom entries” on page 251 explains what the selector value is. You use the <cics:selector> element in the Atom configuration file to specify the way in which the selector value is appended to the path. When you select the default "segment" style or omit

the element, CICS creates links such as <atom:link rel="self" href="/myatomfeed/entries/23">. The alternative "query" style produces a format that is compatible with applications developed for the CA8K SupportPac. You can also use the <cics:selector> element if you need to specify that your selector value is hexadecimal.

In the Atom configuration file, you may omit the scheme and host components of the URL, and specify only the path component. CICS adds the scheme and host components to the URL when it returns the Atom feed or Atom entry document to the client, to comply with the Atom format specification.

<atom:published>
The time when the Atom entry was first created or published. This element is optional. This element corresponds to the published attribute of the <cics:fieldnames> element. If the CICS resource does not provide this data, CICS supplies the current date and time as a default. You can specify the <atom:published> element in your Atom configuration file with an alternative default timestamp in the XML dateTime format, as described in RFC 3339.

<atom:rights>
A text string that contains the claimed intellectual property rights, such as copyright. CICS only supports plain text for this element. This element is optional, and if it is not provided, the <atom:rights> element for the Atom feed applies. The <cics:fieldnames> element has no corresponding attribute, so the data that you provide applies to all Atom entries.

<atom:summary>
A short description of the content of the Atom entry. CICS only supports plain text for summaries. This element corresponds to the summary attribute of the <cics:fieldnames> element. This element is required if the content of the Atom entry is not text, HTML, XHTML or XML, so if you plan to provide any content that does not fit these categories, and the CICS resource does not always provide a summary, use this element to provide a default summary. You can also use this element if the CICS resource does not provide a summary and you want the same summary to appear on all your Atom entries. Otherwise, you can omit the element. CICS checks that you do not specify more than one <atom:summary> element for the Atom entry.

<atom:title>
The title for the Atom entry. CICS only supports plain text for titles. This element corresponds to the title attribute of the <cics:fieldnames> element. You must include an <atom:title> element in the prototype Atom entry, even if your CICS resource always provides a title for Atom entries, to comply with RFC 4287. Use a suitable default title that could apply to any of your Atom entries, or use an empty element if your Atom entries all have a title and no default would be suitable. CICS checks that you do not specify more than one <atom:title> element.

<atom:updated>
The time when the Atom entry was last updated in a significant way. This element is required by the Atom specification, but you cannot specify it in the prototype Atom entry in your Atom configuration file. The element corresponds to the updated attribute of the <cics:fieldnames> element. If the CICS resource does not provide this data, CICS supplies the current date and time as a default.

Example

```
<atom:entry>
    <atom:title>An entry from my feed</atom:title>
    <atom:summary>DEFAULT --- This is the default summary</atom:summary>
    <atom:link rel="self" href="/web20/sample_atom_feed/entry" />
    <atom:author>
          <atom:name>Joe Bloggs</atom:name>
          <atom:uri>http://www.hursley.ibm.com/JBloggs/</atom:uri>
          <atom:email>JBloggs@uk.ibm.com</atom:email>
    </atom:author>
    <atom:contributor>
          <atom:name>John Doe</atom:name>
    </atom:contributor>
    <atom:category term="Comments" />
    <atom:published>2008-12-02T15:41:00</atom:published>
    <atom:content type="text" cics:resource="WB20TSQ" cics:type="tsqueue" />
 </atom:entry>
```

Atom element reference for CICS

These tables provide a reference for the relationships between the elements used in an Atom feed document; the elements used in an Atom entry element; the attributes of the <cics:fieldnames> element; and the parameters that CICS can pass to a service routine for resource handling.

When you create an Atom feed document in CICS, you can specify the elements that are defined by the Atom format specification in RFC 4287. Some of these elements are used as child elements of the <atom:feed> element to supply metadata for the whole of the Atom feed, such as the title for the feed. Some of the elements are used as child elements of an <atom:entry> element to supply metadata or content for an individual entry. The majority of the elements are specified for the feed and specified again for the individual entries; for example, each entry has a unique identifier specified by the <atom:id> element, and the feed also has a unique identifier. For the complete descriptions of each element, read RFC 4287.

In an Atom feed document in CICS, CICS uses a single prototype <atom:entry> element to generate the individual entries. If you specify child elements in this element, the metadata there applies to all entries by default. However, if the CICS resource that you are using to supply the content for the Atom feed contains suitable metadata, you can use attributes of the <cics:fieldnames> element to tell CICS if and where the data for the child elements is present in the resource record. For example, you can specify a field in the resource record that supplies the title for the entry that is contained in the resource record. If your resource records do not contain certain metadata, such as the author's name, you can omit that attribute for the <cics:fieldnames> element. CICS either supplies that item of metadata from the corresponding element in the prototype <atom:entry> element in the configuration file or omits it. If your resource records do not contain any suitable metadata, you can omit the <cics:fieldnames> element completely, and CICS publishes the whole of the resource record as the content of the entry.

The parameters that CICS passes to a service routine include parameters corresponding to the attributes of the <cics:fieldnames> element. You can use these resource handling parameters if you want to write a service routine that obtains its information about resource structures from the Atom configuration file, rather than having this information coded directly in the service routine. With this method, you can create a generic service routine that is capable of handling multiple resources.

Table 13. Child elements of the <atom:feed> and <atom:entry> elements

Element	Meaning	For feed	For entries	<cics:fieldnames> attribute (for entries)	Service routine parameter (for entries)
<app:edited>	Time when entry was last edited	Not used	Required if in collection, produced by CICS	edited	ATMP_ EDITED
<atom:author>	Principal author's details	Required unless all entries have this element	Optional if feed has this element	Not applicable (data is in child elements)	Not applicable (data is in child elements)
<atom:category>	Category classifying feed or entry	Optional	Optional	category	ATMP_ CATEGORY_ FLD
<atom:content type=" ">	Content of entry	Not used	Required	content, content_type	ATMP_ CONTENT_ FLD and ATMP_ CONTENT_ TYPE_FLD
<atom:contributor>	Subsidiary author's details	Optional	Optional	Not applicable (data is in child elements)	Not applicable (data is in child elements)
<atom:email>	Email address of author or contributor	Optional	Optional	email (for author only, contributors not supported)	ATMP_ EMAIL_ FLD
<atom:generator>	Agent that generates the feed	Produced by CICS	Not used	Not applicable (not used for entries)	Not applicable (not used for entries)
<atom:icon>	Icon representing feed	Optional	Not used	Not applicable (not used for entries)	Not applicable (not used for entries)
<atom:id>	Unique identifier for feed or entry	Required, CICS produces if you specify <cics: authority> element	Required, CICS produces if you specify <cics: authority> element	atomid	ATMP_ ID_ FLD
<atom:link rel="self">	URL for retrieving feed or entry document	Required	Required by CICS	Not applicable (not stored in resource)	Not applicable (not stored in resource)
<atom:link rel="edit">	URL for editing entry in a collection (Member URI)	CICS produces for collections	CICS produces if entry is in collection	Not applicable (not stored in resource)	Not applicable (not stored in resource)
<atom:logo>	Logo for feed	Optional	Not used	Not applicable (not used for entries)	Not applicable (not used for entries)

Table 13. Child elements of the <atom:feed> and <atom:entry> elements (continued)

Element	Meaning	For feed	For entries	<cics:fieldnames> attribute (for entries)	Service routine parameter (for entries)
<atom:name>	Name of author or contributor	Required in author or contributor element	Required in author or contributor element	author (for author only, contributors not supported)	ATMP_ AUTHOR_ FLD
<atom:published>	Time when entry was first created or published	Not used	Optional	published	ATMP_ PUBLISHED_ FLD
<atom:rights>	Intellectual property rights for feed	Optional	Optional	Not supported in resource	Not supported in resource
<atom:source>	Metadata from use of the entry in another feed	Not used	Optional, but not supported by CICS	Not supported	Not supported
<atom:subtitle>	Subtitle for feed	Optional	Not used	Not applicable (not used for entries)	Not applicable (not used for entries)
<atom:summary>	Short description of entry content	Not used	Required if content not text or XML	summary	ATMP_ SUMMARY_ FLD
<atom:title>	Title for feed	Required	Required	title	ATMP_ TITLE_ FLD
<atom:updated>	Time when feed was last updated	Required, produced by CICS	Required, produced by CICS	updated	ATMP_ UPDATED_ FLD
<atom:uri>	URL for author or contributor Web site	Optional	Optional	authoruri (for author only, contributors not supported)	ATMP_ AUTHORURI_ FLD

Creating an ATOMSERVICE definition for an Atom feed

Create an ATOMSERVICE definition to identify the Atom configuration file that defines the feed document, the CICS resource that provides the data for the feed, and the XML binding that describes the layout of the resource.

About this task

The CICS resource group DFH$WEB2 contains some sample ATOMSERVICE resource definitions for Atom collections, DFH$W2F1, DFH$W2P1, and DFH$W2Q1. DFH$W2Q1 is used to deliver the sample Atom collection described in the Web 2.0 scenario "Create an Atom feed to work with employee information" in the CICS TS for z/OS, Version 4.1 Information Center, which is available at `https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp`. DFH$W2P1 is used to run the DFH0W2F1 COBOL sample service routine for Atom feeds. DFH$W2F1 is used to deliver the FILEA sample as an Atom feed directly from CICS.

The *CICS Resource Definition Guide* has information about the different methods of resource definition, and full reference information for the ATOMSERVICE resource definition attributes.

Procedure

1. Begin an ATOMSERVICE resource definition with the name that you specified in the URIMAP resource definition for the Atom document, and a group of your choice, using one of the methods listed in the *CICS Resource Definition Guide*.
2. Use the STATUS attribute to specify whether the ATOMSERVICE resource definition will be installed in an enabled or disabled state.
3. Specify an ATOMTYPE attribute of FEED.
4. Specify the CONFIGFILE attribute as the name and path for the Atom configuration file that you created for the Atom document.
5. Use the RESOURCETYPE attribute to specify whether the CICS resource that provides the data for the Atom entries is a service routine (PROGRAM), or a file that CICS handles directly (FILE), or a temporary storage queue that CICS handles directly (TSQUEUE).
6. Specify the RESOURCENAME attribute as the 1-16 character name of the CICS resource.
7. Specify the BINDFILE attribute as the name and path for the XML binding that you created using the CICS XML assistant. An XML binding is required for FILE or TSQUEUE resources. If you are using a service routine (PROGRAM) and you created an XML binding for the resource that holds the data for your Atom entries, specify the XML binding for that resource. If you are using a service routine that holds its own information about the resource structure and you do not have an XML binding, do not specify this attribute.
8. Install the ATOMSERVICE resource definition and the corresponding URIMAP resource definition that you created in "Creating a URIMAP resource definition for an Atom document" on page 295. If you created a TRANSACTION resource definition for an alias transaction in "Creating an alias transaction for an Atom feed" on page 294, install this resource as well.

Results

When you have installed the ATOMSERVICE resource definition and its supporting resources, you have an Atom feed that Web clients can access to obtain a list of entries in the Atom format. The Web clients must process and display the data themselves.

Many free or commercially available Web client applications are able to request, receive and display Atom feeds, including dedicated feed readers and also applications that provide further functions, such as applications for creating mashups. Check that the application is described as supporting the Atom format. You can also write your own Web client application to make GET requests for Atom feed data. For instructions to write your own HTTP GET requests for Atom feeds, see "Making GET requests to Atom feeds or collections" on page 336.

What to do next

You can take further steps to set up your Atom feed as a collection, so that you or others can manage and edit the entries in the feed through a Web client that supports POST, PUT, and DELETE requests for Atom feeds, as described in the Atom Publishing Protocol.

Chapter 24. Making an Atom feed into a collection

To create an editable collection from an existing Atom feed, set up new URIMAP and ATOMSERVICE resource definitions and a new Atom configuration file. Your new definitions use most of the settings from the original Atom feed with some small changes.

Before you begin

If your Atom feed involves a service routine that extracts data from a resource and supplies to CICS, before you make the data available as a collection, read Chapter 25, "How to edit Atom collections," on page 333, and modify your service routine to take appropriate actions for POST, PUT, and DELETE requests as well as GET requests for the entries in the collection, by following the instructions in "How to handle Atom collection editing requests in your service routine" on page 346.

About this task

A client can use a collection in the same way as an ordinary Atom feed, by obtaining lists of the Atom entries and displaying them. It would therefore be possible for you to change the ATOMSERVICE definition and configuration file for your Atom feed to meet the requirements for a collection, and let CICS deliver the collection to all users in place of the original Atom feed. However, good practice is to create separate CICS resource definitions and use separate URLs to make your Atom entries available as a collection, and continue to make them available separately as a feed. Setting up the same data as a feed and as a collection does involve additional work, but has some important advantages:

- You can apply appropriate security measures to the editable collection, and make the read-only feed freely available.
- You might achieve better response times by delivering the better-performing feed document to most users. Delivering entries from a collection requires more processing than delivering entries from a feed, because of the extra navigation that CICS provides for a collection.

To create an editable collection from an Atom feed:

Procedure

1. Plan appropriate security measures for your collection, so that you allow only authenticated Web clients to edit the entries in the collection. For more information about security, see Chapter 26, "Security for Atom feeds," on page 357.
2. Set up a TCPIPSERVICE resource definition to specify the security measures that are applied for the port where Web clients make requests for your collection. "Creating TCPIPSERVICE resource definitions for CICS Web support" on page 96 explains how to do this.
3. Following the procedure in "Creating a URIMAP resource definition for an Atom document" on page 295, select a suitable URL for your collection that is different from the URL that you used for the Atom feed, and create a new URIMAP definition for the URL of the collection. As well as being different

from the URL for the Atom feed, the URL that you choose must also be different from the URL for other Atom feeds and collections that you serve using the same host name.

4. Complete the steps in "Creating an ATOMSERVICE definition and Atom configuration file for a collection" to set up a new ATOMSERVICE definition and Atom configuration file based on your existing files for the Atom feed. If you are using resource and command security to protect your collection, make sure that the user IDs of Web clients have access to the ATOMSERVICE definition and the resources that it references, including any CICS resources and commands used by a service routine.
5. Create an Atom service document that includes the collection, following the instructions in "Creating an Atom service document" on page 323. You can also create an Atom category document to specify categories for your collection, following the instructions in "Creating an Atom category document" on page 328.
6. Set up CICS resource definitions to deliver your Atom service and category documents, following the instructions in "Delivering an Atom service or category document as an Atom configuration file" on page 330 or "Delivering an Atom service or category document as a static response" on page 331.

Results

When you have completed these tasks, your collection is available for Web clients to add, update, and delete entries. Web clients can discover the URL for your collection by obtaining the service document.

What to do next

Chapter 25, "How to edit Atom collections," on page 333 explains how you can edit your collection by issuing HTTP GET, POST, PUT, and DELETE requests through a Web client, and how a service routine must handle editing requests.

Creating an ATOMSERVICE definition and Atom configuration file for a collection

Create an ATOMSERVICE definition and Atom configuration file for a collection by copying the equivalent files for the Atom feed from the same data and making minor changes.

About this task

The *CICS Resource Definition Guide* has information about the different methods of resource definition and full reference information for the ATOMSERVICE resource definition attributes.

Procedure

1. Copy the Atom configuration file for the Atom feed and rename it with any suitable name of your choice. Make the following changes to the file:
   a. Change the root element from `<cics:atomservice type="feed">` to `<cics:atomservice type="collection">`.
   b. Change the <atom:title> child element of the <atom:feed> element to a suitable title for the collection. You do not have to change the <atom:title> child element of the <atom:entry> element.

c. Change the <atom:id> child element of the <atom:feed> element to a suitable unique identifier for the collection.

d. Change the href attribute of the <atom:link rel="self" href=" "> child element of the <atom:feed> element to specify the complete path that Web clients can use to retrieve the collection. The beginning of the path must match the partial path that you stated in the URIMAP resource definition for the collection. For example, if you specified `/myatomcoll/*` as the path component in the URIMAP resource definition, you could specify <atom:link rel="self" href="/myatomcoll/collection.atom"> in the Atom configuration file. You may omit the scheme and host components of the URL, and specify only the path component.

e. In the <atom:link rel="self" href=" "> child element of the prototype <atom:entry> element, change the href attribute so that the beginning of this standard path matches the partial path that you stated in the URIMAP resource definition for the collection. It is helpful to users (but not required) to also change the remainder of the path so that it is different from the path for the ordinary Atom feed from the corresponding data. The whole path must be different from the complete path that you specified in the <atom:link rel="self" href=" "> element for the collection. CICS appends the selector value supplied by the service routine, or a suitable unique identifier for the Atom entry, to the path to create a complete link for each Atom entry.

   For example, if you specified `/myatomcoll/*` as the path component in the URIMAP resource definition, and <atom:link rel="self" href="/myatomcoll/collection.atom"> as the link for the whole collection in the Atom configuration file, you could specify <atom:link rel="self" href="/myatomcoll/edit/"> as the standard path for entries in the collection. When CICS issues the Atom entry document, CICS changes the "self" attribute to "edit" as required by the Atom Publishing Protocol. Do not change this attribute yourself in the Atom configuration file. In this example, CICS issues links such as <atom:link rel="edit" href="/myatomcoll/edit/23">.

2. Copy the ATOMSERVICE resource definition for the Atom feed, and rename it using the name that you specified in the URIMAP resource definition for the collection. Make the following changes to the definition:

   a. Change the ATOMTYPE attribute from FEED to COLLECTION.

   b. Change the CONFIGFILE attribute to the name of the Atom configuration file that you have just created for the collection.

3. Install the ATOMSERVICE resource definition and the corresponding URIMAP resource definition that you created for the collection. You can also create a TRANSACTION resource definition for an alias transaction for the collection, following the instructions in “Creating an alias transaction for an Atom feed” on page 294, and install this resource definition.

Creating an Atom service document

Create an Atom service document to inform clients about the collections that are available from your server. An Atom service document lists only Atom feeds that you want to make available as collections for editing; it does not include ordinary Atom feeds that are not available for editing.

About this task

You normally create only one Atom service document for the collections that are available through a CICS region. The Atom service document is stored in z/OS UNIX System Services. The Atom service document is an XML document, and the file has the .xml extension. You can create the file using any XML editor or text editor.

You can choose whether to create your Atom service document as an Atom configuration file, which CICS manages using an ATOMSERVICE resource definition, or as a file delivered by CICS Web Support static content delivery. Defining the Atom service document as an Atom configuration file requires an extra resource definition, but means that the document is integrated with CICS support for Atom feeds.

If you choose to define the Atom service document as an Atom configuration file, you begin the document with the root element <cics:atomservice type="service">. Other than that element, the Atom service document for your CICS collections does not contain any elements that are specific to the CICS environment. It uses the standard elements that are defined by the Atom Publishing Protocol specification in RFC 5023. The CICS documentation therefore does not include further reference information for these elements. If you need any further reference information for the elements in an Atom service document, consult RFC 5023, *The Atom Publishing Protocol*.

CICS does not validate the content of your Atom service document. If you correctly follow the steps listed here, you can produce a valid Atom service document that is compliant with the Atom Publishing Protocol specification. If you find that clients experience any problems reading your service document, or do not interpret it in the way that you expected, recheck your service document against these instructions and against the Atom Publishing Protocol specification in RFC 5023.

Procedure

1. If you want to define your Atom service document as an Atom configuration file, begin the document with the root element <cics:atomservice type="service">, and add the <app:service> element as a child element.

   ```
   <?xml version="1.0"?>
   <cics:atomservice type="service">
    <app:service>
    </app:service>
   </cics:atomservice>
   ```

   If you do not want to define the Atom service document as an Atom configuration file, use the <app:service> element as the root element, as follows:

   ```
   <?xml version="1.0"?>
   <app:service>
   </app:service>
   ```

2. Include the namespace declarations for the Atom XML namespace and the namespace for the Atom Publishing Protocol in the root element of the Atom service document; that is, either the <cics:atomservice> element or the <app:service> element. If you have used the <cics:atomservice> element as the root element, also include the namespace declaration for the CICS Atom XML namespace. For example:

```
<?xml version="1.0"?>
<cics:atomservice type="service"
      xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
 <app:service>
 </app:service>
</cics:atomservice>
```

With the <app:service> element as the root element, the declarations look like this:

```
<?xml version="1.0"?>
<app:service
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
</app:service>
```

3. Add at least one <app:workspace> element as a child element of the <app:service> element, and add an <atom:title> element to each workspace with a suitable title. <app:workspace> elements just group collections together in the service document, and the server and client do not have to take any actions relating to them. However, the Atom Publishing Protocol requires that you use at least one workspace, and that each workspace has a human-readable title. This example shows a single workspace that can contain all your collections:

```
<?xml version="1.0"?>
<cics:atomservice type="service"
      xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
 <app:service>
  <app:workspace>
     <atom:title>CICS Atom collections</atom:title>
  </app:workspace>
 </app:service>
</cics:atomservice>
```

4. For each of your collections, add an <app:collection> element as a child element of the <app:workspace> element (or in a suitable <app:workspace> element if you have created more than one). Specify the following required items in each <app:collection> element:
   a. An href attribute of the <app:collection> element, stating the complete URL for the collection, with the complete path that you specified in the Atom configuration file for the collection. Web clients use this URL for GET requests to retrieve a list of the Atom entries in the collection. They also use this URL for POST requests to submit new Atom entries to the collection. You must ensure that this link is valid, and that you have set up all the items needed to support the collection, including a URIMAP resource definition and an ATOMSERVICE resource definition.
   b. An <atom:title> element as a child element, stating the title of the collection.

   For example:

```
<app:workspace>
    <atom:title>CICS Atom collections</atom:title>
  <app:collection
      href="http://www.example.com/customers/customercol.atom">
         <atom:title>Customer collection</atom:title>
  </app:collection>
</app:workspace>
```

5. Optional: If you do *not* want clients to create new Atom entries in one or more of your collections, include an empty <app:accept> element as a child element of the <app:collection> element. For example:

```
<app:collection
    href="http://www.example.com/customers/customercol.atom">
       <atom:title>Customer collection</atom:title>
       <app:accept/>
</app:collection>
```

   You must implement additional security measures to prevent clients from editing the collection. Clients should interpret an empty <app:accept> element to mean that they cannot create entries, but the presence of an empty <app:accept> element does not make CICS prevent a client from attempting to create an entry.

   If you want clients to create Atom entries, omit the <app:accept> element. Because CICS does not support media resources, do not use the <app:accept> element to specify any additional media types for entries. CICS rejects any client request to create an entry that is not an Atom entry document; that is, a document with the media type application/atom+xml, with or without the type=entry attribute. This media type is the default, so when you omit the <app:accept> element, clients should understand that they can create only Atom entry documents in this collection.

6. Optional: If you want to specify categories (which are optional) for the Atom entries in one or more of your collections, decide whether you want to provide the list of categories in the service document itself, or supply a reference to a separate category document. A separate category document is useful if you have a long list of categories, or if you want to reuse the same list of categories for different collections. The Atom Publishing Protocol permits you to use multiple <app:categories> elements in a service document, so you might choose, for example, to provide both a reference to a shared Atom category document and a list of categories specific to the collection. When you have made your decision, proceed as follows:

   a. To provide categories in the service document, add an <app:categories> element as a child element of the <app:collection> element, then add one or more <atom:category> elements as child elements of the <app:categories> element. Give each <atom:category> element a term attribute that specifies the name of the category. CICS does not support the optional scheme and label attributes, so do not use these attributes. For example:

```
<app:collection
    href="http://www.example.com/customers/customercol.atom">
       <atom:title>Customer collection</atom:title>
    <app:categories>
       <atom:category term="Customers" />
       <atom:category term="Actions" />
    </app:categories>
</app:collection>
```

      The categories that you specify do not affect the way CICS behaves. CICS accepts client requests that specify a category that is not included in your list. For this reason, for a request where CICS handles the resource directly, do not use the fixed="yes" attribute in your <app:categories> element, which indicates that the server does not allow any other categories. When you omit the fixed attribute, a value of "no" is assumed. If you are using a service routine to make changes to the resource, you may code your service routine to reject client requests on the basis of categories, and indicate this in your <app:categories> element.

b. To provide categories in a separate category document, follow the instructions in "Creating an Atom category document" on page 328 to create a category document and the resource definitions that it requires. Then add an <app:categories> element as a child element of the <app:collection> element, and give it an href attribute specifying the URL that the client can use to retrieve your category document. For example:

```
<app:collection
    href="http://www.example.com/customers/customercol.atom">
       <atom:title>Customer collection</atom:title>
    <app:categories href="http://www.example.com/cat/customercol"/>
</app:collection>
```

You must not use any attributes or child elements in the <app:categories> element when you specify a reference to a category document.

Example Atom service document with two collections

This Atom service document is defined as an Atom configuration file, with the root element <cics:atomservice type="service">. It contains a single workspace with two collections. One of the collections has a list of categories in the service document. The other collection has a reference to a separate category document.

```
<?xml version="1.0"?>
<cics:atomservice type="service"
      xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
 <app:service>
  <app:workspace>
    <atom:title>CICS Atom collections</atom:title>
   <app:collection
      href="http://www.example.com/customers/customercol.atom">
         <atom:title>Customer collection</atom:title>
      <app:categories>
         <atom:category term="Customers" />
         <atom:category term="Actions" />
      </app:categories>
   </app:collection>
   <app:collection
      href="http://www.example.com/sysadmin/messagecol.atom">
      <atom:title>Messages from the systems administrator</atom:title>
      <app:categories href="http://www.example.com/cat/messagecol"/>
   </app:collection>
  </app:workspace>
 </app:service>
</cics:atomservice>
```

What to do next

The Atom Publishing Protocol permits you to specify an <app:collection> element in an Atom feed document, as a child element of the <atom:feed> element. This arrangement can be a useful way to link a collection to the ordinary Atom feed for that data. Clients that understand this markup can see that the Atom feed they requested is available as a collection, and can use the URL to create new entries or browse the entries for editing, without having to view the service document.

If you want to specify your <app:collection> element in an Atom feed document, copy your <app:collection> element and all its child elements from your Atom service document into the Atom configuration file for the Atom feed, as a child element of the <atom:feed> element. Make sure that it is a child of that element, and not of the <cics:feed> element or of the prototype <atom:entry> element. You

must also keep the specification of the <app:collection> element in the Atom service document. If in the future you change the specification of the <app:collection> element in the Atom service document, make corresponding changes in the copy in the Atom configuration file.

Next, follow the instructions in either "Delivering an Atom service or category document as an Atom configuration file" on page 330 or "Delivering an Atom service or category document as a static response" on page 331 to set up the resource definitions to deliver your Atom service document to Web clients.

Creating an Atom category document

Create an Atom category document if you want to supply a long list of categories for a collection or to reuse the same list of categories for different collections. For short or unique lists of categories, there is little value in defining a separate category document; specify the categories in the Atom service document instead.

About this task

An Atom category document is stored in z/OS UNIX System Services. An Atom category document is an XML document, and the file has the .xml extension. You can create the file using any XML editor or text editor.

As for an Atom service document, you can choose whether to create your Atom category documents as Atom configuration files, or as files delivered by CICS Web Support static content delivery. Choose the same approach as you took for your Atom service document in "Creating an Atom service document" on page 323. CICS does not validate the content of an Atom category document.

Apart from the <cics:atomservice type="category"> root element, if you use it, an Atom category document in CICS uses only the standard elements defined in RFC 5023, *The Atom Publishing Protocol*. Consult that document if you need any further reference information.

The categories that you specify in a category document do not affect the way CICS behaves. CICS accepts client requests that specify a category that is not included in your document. For this reason, for a request where CICS handles the resource directly, do not use the fixed="yes" attribute in your <app:categories> element, which indicates that the server does not allow any other categories. If you are using a service routine to make changes to the resource, you may code your service routine to reject client requests on the basis of categories, and indicate this in your <app:categories> element.

Note that for data held in a resource, CICS supports only a single category for each Atom entry.

Procedure

1. If you want to define your Atom category document as an Atom configuration file, begin the document with the root element <cics:atomservice type="category">, and add the <app:categories> element as a child element.

```
<?xml version="1.0"?>
<cics:atomservice type="category">
 <app:categories>
 </app:categories>
</cics:atomservice>
```

If you do not want to define the Atom category document as an Atom configuration file, use the <app:categories> element as the root element, as follows:

```
<?xml version="1.0"?>
<app:categories>
</app:categories>
```

Do not use the optional scheme attribute, because CICS does not support it. For a request where CICS handles the resource directly, do not use the fixed="yes" attribute, because CICS does not provide any support for disallowing categories. When you omit the fixed attribute, a value of "no" is assumed. For a request where you are using a service routine to handle the resource, you may use the fixed="yes" attribute if your service routine rejects requests on the basis of categories.

2. Include the namespace declarations for the Atom XML namespace and the namespace for the Atom Publishing Protocol in the root element of the Atom category document; that is, either the <cics:atomservice> element or the <app:categories> element. If you have used the <cics:atomservice> element as the root element, also include the namespace declaration for the CICS Atom XML namespace. For example:

```
<?xml version="1.0"?>
<cics:atomservice type="category"
      xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
 <app:categories>
 </app:categories>
</cics:atomservice>
```

With the <app:categories> element as the root element, the declarations look like this:

```
<?xml version="1.0"?>
<app:categories
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
</app:categories>
```

3. Add one or more <atom:category> elements as child elements of the <app:categories> element, and give each <atom:category> element a term attribute that specifies the name of the category. For example:

```
 <app:categories>
    <atom:category term="Events" />
    <atom:category term="Comments" />
 </app:categories>
```

CICS does not support the optional scheme and label attributes, so do not use these attributes.

4. Include the full URL for your category document in your Atom service document, by specifying the <app:categories> element with the href attribute in each applicable collection. This example shows how to specify the <app:categories> element as a child of the <app:collection> element in an Atom service document:

```
  <app:collection
      href="http://www.example.com/events/eventcol.atom">
         <atom:title>Events collection</atom:title>
      <app:categories href="http://www.example.com/cat/eventcol"/>
  </app:collection>
```

Example

This Atom category document is defined as an Atom configuration file, with the root element <cics:atomservice type="category">.

```
 <?xml version="1.0"?>
<cics:atomservice type="category"
      xmlns:cics="http://www.ibm.com/xmlns/prod/cics/atom/atomservice"
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app">
 <app:categories>
    <atom:category term="Events" />
    <atom:category term="Comments" />
 </app:categories>
</cics:atomservice>
```

What to do next

Follow the instructions in either "Delivering an Atom service or category document as an Atom configuration file" or "Delivering an Atom service or category document as a static response" on page 331 to set up the resource definitions to deliver your Atom category document to Web clients.

Delivering an Atom service or category document as an Atom configuration file

If you created your Atom service or category document as an Atom configuration file with the root element <cics:atomservice>, set up URIMAP and ATOMSERVICE resource definitions to deliver the document.

Procedure

1. Set up a URIMAP definition for the Atom service or category document following the instructions in "Creating a URIMAP resource definition for an Atom document" on page 295. The path component of the URL that you choose for the Atom service document must **not** begin with any of the common parts of path components that you specify in URIMAP resource definitions for Atom feeds or collections that you serve using the same host name.
2. Begin an ATOMSERVICE resource definition with the name that you specified in the URIMAP resource definition for the Atom document, and a group of your choice, using one of the methods listed in the *CICS Resource Definition Guide*.
3. Use the STATUS attribute to specify whether the ATOMSERVICE resource definition will be installed in an enabled or disabled state.
4. Specify an ATOMTYPE attribute of SERVICE for an Atom service document, or CATEGORY for an Atom category document.
5. Specify the CONFIGFILE attribute as the name and path for the Atom configuration file that you created for the Atom document.
6. Install the ATOMSERVICE resource definition and the corresponding URIMAP resource definition that you created in "Creating a URIMAP resource definition for an Atom document" on page 295. If you created a TRANSACTION resource definition for an alias transaction in "Creating an alias transaction for an Atom feed" on page 294, install this resource as well.

Delivering an Atom service or category document as a static response

If you created a plain Atom service or category document with the root element <app:service> or <app:categories>, set up a URIMAP resource definition to deliver the document as a static response through CICS Web support.

Procedure

1. Complete the steps for planning to deliver a z/OS UNIX file as a static response, as described in "Providing static HTTP responses with a CICS document template or z/OS UNIX file" on page 64.
2. Complete the steps for setting up a URIMAP definition for a z/OS UNIX file as a static response, as described in "Starting a URIMAP resource definition for any requests for CICS as an HTTP server" on page 100 and "Completing a URIMAP definition for a static response to an HTTP request for CICS as an HTTP server" on page 103. When you are following these steps, make these choices in the URIMAP definition for your Atom service or category document:
   a. Specify the PATH attribute as a suitable URL for the service or category document, such as `/servicedocument`. The path for an Atom service or category document must not begin with the common part of the path component for any of the Atom feeds and collections that you serve using the host name that you specified.
   b. Remember to specify a USAGE attribute of SERVER (CICS as an HTTP server), not ATOM.
   c. Specify the MEDIATYPE attribute as `application/atomsvc+xml` for an Atom service document, or `application/atomcat+xml` for an Atom category document.
   d. Specify the HFSFILE attribute as the name of the z/OS UNIX file that contains your service or category document.

Chapter 25. How to edit Atom collections

When Atom entries are made available as a collection, a client can edit or delete the existing entries and create new entries for the collection.

To discover which collections are available on a server, the client requests a service document from the server. The service document lists the URLs of the collections that are available to the client. The client can then interact with the Atom entries by sending HTTP requests to the server as follows:

GET Retrieve a single Atom entry or a list of Atom entries. GET requests for a list of Atom entries are sent to the URL of the collection, as stated in the Atom service document. GET requests for a single Atom entry are sent to the URL of an individual Atom entry in the collection, as stated in the <atom:link rel="edit"> link for the entry.

POST Create a new Atom entry. POST requests are sent to the URL of the collection.

PUT Edit an existing Atom entry that the client has obtained using a GET request. PUT requests are sent to the URL of an individual Atom entry in the collection.

DELETE
Delete an existing Atom entry. DELETE requests are sent to the URL of an individual Atom entry in the collection.

For POST and PUT requests, the client sends a request body containing a complete Atom entry document. (CICS does not support other media types in Atom collections.) Although a PUT request typically updates only a part of an existing entry, the client should send the whole entry, including the elements of the existing entry that were not edited, to avoid any possible misinterpretation by the server. The server adds the new entry or updates the existing entry, then sends an HTTP response to the client with an appropriate HTTP status code. The server can change, add to, or remove the items of metadata (such as the Atom ID or the date and time stamps) that the client provides for an entry. So when a client makes a successful POST or PUT request, the server also returns a copy of the new entry as the body of the response. CICS requires entity tags (ETags) for PUT requests for Atom entries in a collection, which enable the server to confirm that the client has based its editing requests on an up-to-date copy of the Atom entry.

RFC 5023, *The Atom Publishing Protocol*, which is available from http://www.ietf.org/rfc/rfc5023.txt, has a full statement of the protocol for interacting with Atom entries in a collection, and some examples of the HTTP requests and responses used.

For a collection for which CICS extracts data directly from a file or temporary storage queue without the involvement of a service routine, CICS carries out the actions of a server to process the client request, update the resource, and return a response to the client. Be aware that these actions modify the contents of the file or temporary storage queue permanently. CICS adds or edits records in response to POST and PUT requests. For DELETE requests relating to a file, CICS deletes the records, except in an ESDS where records cannot be deleted. For DELETE requests relating to a temporary storage queue, CICS removes the records by setting the

first byte to 'FF'x. If these actions are not appropriate for your environment, use a service routine instead to handle the client requests.

CICS does not support some of the possible actions described for collections in the Atom Publishing Protocol (RFC 5023), in particular the following:

- CICS does not support media resources and media link entries in collections. Media resources are specified by the Atom Publishing Protocol (RFC 5023) primarily as a means of organizing nontext content in a collection. When a client attempts to create an entry in a collection, CICS rejects with a 415 status code any client request that is not an Atom entry (with the media type application/atom+xml, with or without the type=entry parameter). Do not specify any additional media types in <app:accept> elements in a service document for CICS.
- CICS does not reject Atom entries for a collection on the basis of categories. You can use the <app:categories> element in a service document or category document to specify acceptable categories for entries in a collection, but CICS does not police whether clients adhere to these categories.
- For reasons of performance, CICS does not automatically return Atom entries in a collection in the order in which they were most recently edited (as shown by the <app:edited> element in the entry). This function is a SHOULD requirement in RFC 5023 for a full list of entries, but a MUST requirement for a partial list of entries. CICS deviates from this requirement in order to maintain acceptable response times while still providing the useful function of partial lists. If you are using a service routine to supply the entries to CICS, you can make the collection compliant by supplying the entries in the order in which they were last edited, if your resource is able to store this information.

"Atom features not supported by CICS" on page 48 lists other unsupported items that are minor or do not relate specifically to collections.

For a collection served by CICS involving data that a service routine extracts from a resource and supplies to CICS, CICS passes the client request to the service routine in a set of containers. You must code your service routine to apply the request to the data in the resource and then return a response to CICS to send to the client. In this situation, you share the responsibility with CICS in some respects for compliance with the Atom Publishing Protocol (RFC 5023). "How to handle Atom collection editing requests in your service routine" on page 346 explains how your service routine can handle GET, POST, PUT, and DELETE requests for collections.

When you have finished setting up your Atom collection and (if necessary) your service routine, you can edit the entries using any suitable client that handles the HTTP protocol. The exact process for making the requests and viewing the responses varies depending on the client that you have chosen. For more information about how to interact with the entries in a collection using GET, POST, PUT, and DELETE requests, see "Editing Atom collections using a Web client" on page 335.

Chapter 24, "Making an Atom feed into a collection," on page 321
To create an editable collection from an existing Atom feed, set up new URIMAP and ATOMSERVICE resource definitions and a new Atom configuration file. Your new definitions use most of the settings from the original Atom feed with some small changes.

Editing Atom collections using a Web client

Use a Web client that can make HTTP GET, POST, PUT, and DELETE requests to read, create, edit, and delete entries in your Atom collection.

Before you begin

CICS provides a sample Atom collection supported by a mashup Web page that lets you make Web client requests for the collection, and view the requests and responses. To see how Web clients can edit Atom collections in CICS, set up and use the sample Atom collection following the instructions in the Web 2.0 scenario "Create an Atom feed to work with employee information" in the CICS TS for z/OS, Version 4.1 Information Center, which is available at `https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp`.

About this task

Many free or commercially available Web client applications can make HTTP GET requests to obtain and display Atom feeds or collections. However, not all these Web clients can also make HTTP POST, PUT, and DELETE requests to edit the Atom entries. Check that the application is specifically described as supporting the Atom Publishing Protocol, not just the Atom format. If you have a Web client with this capability, consult the documentation for the client to read about the process for making the requests and viewing the responses. The Web client can interact with CICS as it does with any server that supports the Atom Publishing Protocol, with the exception of certain functions listed in Chapter 25, “How to edit Atom collections,” on page 333.

If you do not have a Web client application that specifically supports POST, PUT, and DELETE requests for Atom entries, you can use a Web client application that lets you compose and send your own HTTP requests and view the responses. You can also write your own Web client applications to make POST, PUT, and DELETE requests to Atom collections. For instructions to make Web client requests from a CICS application, see “Making HTTP requests through CICS as an HTTP client” on page 148.

If you are writing your own Web client application to edit an Atom collection, or if you are composing your own HTTP requests in a suitable Web client, make the requests following the steps described here. If you want further information at any step, see RFC 5023, *The Atom Publishing Protocol*, which is available from http://www.ietf.org/rfc/rfc5023.txt, for details of the protocol for interacting with Atom entries in a collection. RFC 2616, *Hypertext Transfer Protocol -- HTTP/1.1* , which is available from http://www.ietf.org/rfc/rfc2616.txt, describes the protocol for making HTTP requests.

Your collection should have security measures in place to control Web client access. These instructions assume that the Web client that you are using has full access to read and modify the Atom entries. For Atom feeds and collections served from CICS, you can allow some Web clients to have read-only access to the items so that they can make only HTTP GET requests, but other Web clients can have UPDATE access so that they can also make HTTP POST, PUT, and DELETE requests. If your Web client does not have the correct access to carry out an action on the collection, CICS returns an HTTP error response with the status code 403 (Forbidden). For more information about security for Atom feeds and collections, see Chapter 26, “Security for Atom feeds,” on page 357.

Procedure

1. If you do not know the URL of the collection that you want to edit, or you are writing an application that can handle multiple collections, first make an HTTP GET request for the Atom service document that you set up in "Creating an Atom service document" on page 323. The Atom service document lists the URLs of the collections that are available on the server. You can also check the possible categories for entries in the collection, which are listed either in the service document or in a separate category document.
2. To obtain a list of the existing Atom entries in the collection, make an HTTP GET request to the URL of the collection, following the instructions in "Making GET requests to Atom feeds or collections." These instructions also apply to GET requests to an Atom feed that is not defined as a collection.
3. To obtain an individual Atom entry from the collection, make an HTTP GET request to the URL of the Atom entry, following the instructions in "Making GET requests to Atom feeds or collections." These instructions also apply to GET requests to an Atom feed that is not defined as a collection.
4. To create a new Atom entry in the collection, make an HTTP POST request to the URL of the collection, following the instructions in "Making POST requests to Atom collections" on page 341.
5. To edit an existing Atom entry in the collection, make an HTTP PUT request to the URL of the Atom entry, following the instructions in "Making PUT requests to Atom collections" on page 344.
6. To delete an existing Atom entry from the collection, make an HTTP DELETE request to the URL of the Atom entry, following the instructions in "Making DELETE requests to Atom collections" on page 346

Making GET requests to Atom feeds or collections

A Web client can obtain a list of the existing Atom entries in an Atom feed or collection by making an HTTP GET request to the URL of the collection, or obtain an individual Atom entry from the Atom feed or collection by making an HTTP GET request to the URL of the Atom entry.

Before you begin

If you want to edit a collection and you do not know the URL of the collection that you want to edit, or you are writing an application that can handle multiple collections, first make an HTTP GET request for the Atom service document that you set up in "Creating an Atom service document" on page 323. The Atom service document lists the URLs of the collections that are available on the server. The URLs for Atom feeds are not available through Atom service documents, but are typically publicized in a relevant location, for example, as an invitation on a Web site to subscribe to a feed.

Procedure

1. To obtain a list of the existing Atom entries in the Atom feed or collection, make an HTTP GET request to the URL of the Atom feed or collection. Compose the request as follows:
   a. Begin with a request line consisting of the GET method, followed by the path component of the URL of the Atom feed or collection, followed by `HTTP/1.1`, which is the HTTP version for the request. You may also include the scheme (HTTP or HTTPS) and the host name in the URL. For an explanation of request lines, see "HTTP requests" on page 12.

b. If you want to obtain more or less than the default number of Atom entries that CICS sends for a single request for this Atom feed or collection, specify a query string at the end of the URL, with the name `w` (for "window") and the value as the number of Atom entries that you want to receive in this Atom document. This query string requests six Atom entries:

   ```
   ?w=6
   ```

c. Write HTTP headers for the request as follows, each on a new line:

   - The Host header, giving the host name from the URL of the Atom feed or collection, if you did not already include the host name in the request line.
   - The Authorization header, with any security information required to access the Atom feed or collection, such as a user ID and password for basic authentication.

   Put an additional carriage return line feed (CRLF) after the last HTTP header to give an empty line.

Do not include a request body. Send the request to the server. The server returns an Atom feed document, which is a single HTTP response that contains a complete or partial list of the Atom entries in the Atom feed or collection.

Note: The elements in Atom documents served by CICS typically do not include the atom: namespace prefix. The Atom namespace is defined as the default namespace in the element at the beginning of the Atom document, so the atom: prefix is not required for the child elements. However, in references to the elements in the CICS documentation, the atom: prefix is used in the element names for clarity.

2. If you received a partial list of the Atom entries in the Atom feed or collection, and you want to obtain more, make further HTTP GET requests to the links in the Atom feed document that specify further partial lists, as follows:
   a. Use the URL stated in the <atom:link rel="next"> element to obtain the next window or partial list of entries. CICS provides this link for both Atom feeds and collections. As noted above, CICS does not include the atom: namespace prefix for child elements, so the element appears in the Atom feed document as <link rel="next">.
   b. Use the URL stated in the <atom:link rel="previous"> element to obtain the previous partial list of entries. CICS provides this link for collections.
   c. Use the URL stated in the <atom:link rel="first"> element to obtain the first partial list of entries. CICS provides this link for collections.
   d. Use the URL stated in the <atom:link rel="last"> element to obtain the last partial list of entries. CICS provides this link for collections. For Atom documents served by CICS, to improve performance, this partial list contains only the last Atom entry in the feed. You can use the <atom:link rel="previous"> links to retrieve all the previous partial lists.

   Be aware that the links to further partial lists represent a snapshot of the Atom feed or collection at the point in time when CICS issues the Atom document containing them. When Atom entries are added to or deleted from the collection, the links might become invalid. The links are therefore only useful in the short term, to browse a collection that does not change quickly.
3. To obtain an individual Atom entry from the Atom feed or collection, make an HTTP GET request to the URL of the Atom entry, as stated in the <atom:link rel="self"> or <atom:link rel="edit"> element of the entry. <atom:link rel="self"> provides the link for an Atom entry in a feed, and <atom:link rel="edit">

provides the link for an Atom entry in a collection. Atom entries in a collection might also have an <atom:link rel="self"> link. Compose the request as follows:

a. Begin with a request line consisting of the GET method, followed by the path component of the URL of the Atom entry, followed by `HTTP/1.1`, which is the HTTP version for the request. You may also include the scheme (HTTP or HTTPS) and the host name in the URL. For an explanation of request lines, see "HTTP requests" on page 12.

b. Write HTTP headers for the request as follows, each on a new line:

   - The Host header, giving the host name from the URL of the Atom feed or collection, if you did not already include the host name in the request line.
   - The Authorization header, with any security information required to access the Atom feed or collection, such as a user ID and password for basic authentication.

   Put an additional carriage return line feed (CRLF) after the last HTTP header to give an empty line.

Do not include a request body. Send the request to the server. The server returns an HTTP response containing a copy of the individual Atom entry.

Example

This HTTP request is for an Atom collection with the URL `http://www.example.com:80/web20/myfeed` :

```
GET /web20/myfeed HTTP/1.1
Host: www.example.com:80
```

This HTTP request is for an Atom entry with the URL `http://www.example.com:80/web20/entry/7`:

```
GET /web20/entry/7 HTTP/1.1
Host: www.example.com:80
```

This example HTTP response shows an Atom document that CICS sends in response to the request for a single Atom entry. The example shows only the HTTP headers that are of interest for Atom feeds; further HTTP headers might be present in the response. The ETag header for the HTTP response gives the entity tag for the Atom entry, which you must use in the If-Match header if you make a PUT request to edit the entry.

```
HTTP/1.1 200 OK
Content-Type: application/atom+xml
Content-Length: 1005
ETag: c4826af12991fb102ef13099c927c2ac24e4caa2

<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app">
    <generator uri="http://www.ibm.com/cics/" version="6.6.0">
    CICS Transaction Server Version 4.1.0
    </generator>
    <link rel="self" href="http://www.example.com:80/web20/entry/7"/>
    <link rel="edit" href="http://www.example.com:80/web20/entry/7"/>
    <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:7</id>
    <title>This is entry 7</title>
    <summary>
    Entry 7 is about something to do with feeds...
    </summary>
    <category term="Test Feeds"/>
    <rights>Copyright (c) 2009, Joe Bloggs</rights>
    <published>2008-12-02T15:41:00</published>
```

```
|      <author>
|       <name>Joe Bloggs</name>
|       <email>JBloggs@example.com</email>
|       <uri>http://www.example.com/JBloggs/</uri>
|      </author>
|      <contributor>
|       <name>John Doe</name>
|      </contributor>
|      <app:edited>2009-02-02T16:29:36+00:00</app:edited>
|      <updated>2009-02-02T16:29:36+00:00</updated>
|      <content type="text/xml">
|          <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
|          <data_field>
|          Here is some content for entry 7
|          </data_field>
|          </SAMPBIND >
|      </content>
|  </entry>
```

This example HTTP response shows an Atom document that CICS sends in response to the request for a list of Atom entries from the Atom collection. Again, the example shows only the HTTP headers that are of interest for Atom feeds; further HTTP headers might be present in the response.

```
|  HTTP/1.1 200 OK
|  Content-Type: application/atom+xml
|  Content-Length: 8661
|
|  <?xml version="1.0" encoding="utf-8"?>
|  <feed xmlns="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app">
|      <generator uri="http://www.ibm.com/cics/" version="6.6.0">
|      CICS Transaction Server Version 4.1.0
|      </generator>
|      <link rel="self" href="http://www.example.com:80/web20/myfeed"/>
|      <link rel="edit" href="http://www.example.com:80/web20/myfeed"/>
|      <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ</id>
|      <title type="text">CICS ATOM FEED TITLE</title>
|      <subtitle>CICS ATOM FEED SUBTITLE</subtitle>
|      <rights>Copyright (c) 2009, Joe Bloggs</rights>
|      <category term="my-first-feed"/>
|      <author>
|       <name>Joe Bloggs</name>
|       <email>JBloggs@example.com</email>
|       <uri>http://www.example.com/JBloggs/</uri>
|      </author>
|      <contributor>
|       <name>John Doe</name>
|      </contributor>
|      <!--------------------------------------------------------------------->
|      <entry>
|          <link rel="self" href="http://www.example.com:80/web20/entry/9"/>
|          <link rel="edit" href="http://www.example.com:80/web20/entry/9"/>
|          <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:9</id>
|          <title>This is entry 9</title>
|          <summary>
|          Entry 9 is about something to do with feeds...
|          </summary>
|          <category term="Test Feeds"/>
|          <rights>Copyright (c) 2009, Joe Bloggs</rights>
|          <published>2008-12-02T15:41:00</published>
|          <author>
|           <name>Joe Bloggs</name>
|           <email>JBloggs@example.com</email>
|           <uri>http://www.example.com/JBloggs/</uri>
|          </author>
|          <contributor>
|           <name>John Doe</name>
```

```
|          </contributor>
|          <app:edited>2009-02-02T16:29:36+00:00</app:edited>
|          <updated>2009-02-02T16:29:36+00:00</updated>
|          <content type="text/xml">
|           <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
|           <data_field>
|           Here is some content for entry 9
|           </data_field>
|           </SAMPBIND >
|          </content>
|      </entry>
|      <!--------------------------------------------------------------------->
|      <entry>
|          <link rel="self" href="http://www.example.com:80/web20/entry/8"/>
|          <link rel="edit" href="http://www.example.com:80/web20/entry/8"/>
|          <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:8</id>
|          <title>This is entry 8</title>
|          <summary>
|          Entry 8 is about something to do with feeds...
|          </summary>
|          <category term="Test Feeds"/>
|          <rights>Copyright (c) 2009, Joe Bloggs</rights>
|          <published>2008-12-02T15:41:00</published>
|          <author>
|           <name>Joe Bloggs</name>
|           <email>JBloggs@example.com</email>
|           <uri>http://www.example.com/JBloggs/</uri>
|          </author>
|          <contributor>
|           <name>John Doe</name>
|          </contributor>
|          <app:edited>2009-02-02T16:29:36+00:00</app:edited>
|          <updated>2009-02-02T16:29:36+00:00</updated>
|          <content type="text/xml">
|           <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
|           <data_field>
|           Here is some content for entry 8
|           </data_field>
|           </SAMPBIND >
|          </content>
|      </entry>
|      <!--------------------------------------------------------------------->
|      <entry>
|          <link rel="self" href="http://www.example.com:80/web20/entry/7"/>
|          <link rel="edit" href="http://www.example.com:80/web20/entry/7"/>
|          <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:7</id>
|          <title>This is entry 7</title>
|          <summary>
|          Entry 7 is about something to do with feeds...
|          </summary>
|          <category term="Test Feeds"/>
|          <rights>Copyright (c) 2009, Joe Bloggs</rights>
|          <published>2008-12-02T15:41:00</published>
|          <author>
|           <name>Joe Bloggs</name>
|           <email>JBloggs@example.com</email>
|           <uri>http://www.example.com/JBloggs/</uri>
|          </author>
|          <contributor>
|           <name>John Doe</name>
|          </contributor>
|          <app:edited>2009-02-02T16:29:36+00:00</app:edited>
|          <updated>2009-02-02T16:29:36+00:00</updated>
|          <content type="text/xml">
|           <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
|           <data_field>
|           Here is some content for entry 7
```

```
          </data_field>
          </SAMPBIND >
        </content>
    </entry>
    <!--------------------------------------------------------------------->
    <entry>
        <link rel="self" href="http://www.example.com:80/web20/entry/6"/>
        <link rel="edit" href="http://www.example.com:80/web20/entry/6"/>
        <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:6</id>
        <title>This is entry 6</title>
        <summary>
        Entry 6 is about something to do with feeds...
        </summary>
        <category term="Test Feeds"/>
        <rights>Copyright (c) 2009, Joe Bloggs</rights>
        <published>2008-12-02T15:41:00</published>
        <author>
         <name>Joe Bloggs</name>
         <email>JBloggs@example.com</email>
         <uri>http://www.example.com/JBloggs/</uri>
        </author>
        <contributor>
         <name>John Doe</name>
        </contributor>
        <app:edited>2009-02-02T16:29:36+00:00</app:edited>
        <updated>2009-02-02T16:29:36+00:00</updated>
        <content type="text/xml">
         <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
         <data_field>
         Here is some content for entry 6
         </data_field>
         </SAMPBIND >
        </content>
    </entry>
</feed>
```

Making POST requests to Atom collections

A Web client can create a new Atom entry in a collection by making an HTTP POST request to the URL of the collection.

Before you begin

If you do not know the URL of the collection that you want to edit, or you are writing an application that can handle multiple collections, first make an HTTP GET request for the Atom service document that you set up in "Creating an Atom service document" on page 323. The Atom service document lists the URLs of the collections that are available on the server.

Procedure

1. Begin your HTTP POST request with a request line consisting of the POST method, followed by the path component of the URL of the collection, followed by `HTTP/1.1`, which is the HTTP version for the request. You may also include the scheme (HTTP or HTTPS) and the host name in the URL. For an explanation of request lines, see "HTTP requests" on page 12.
2. Write HTTP headers for the request as follows, each on a new line:
   - The Host header, giving the host name from the URL of the collection, if you did not already include the host name in the request line.
   - The Authorization header, with any security information required to access your collection, such as a user ID and password for basic authentication.
   - The Content-Type header, with the value `application/atom+xml;type=entry`.

- The Content-Length header, stating the length of your message body in bytes (octets). Fill in the value for this header when you have finished writing the message body.

Put an additional carriage return line feed (CRLF) after the last HTTP header to give an empty line.

3. Set up a message body containing the XML markup for the Atom entry that you want to post.
   a. If you already have Atom entries in your collection, make a GET request to obtain a feed or entry document for the collection, and copy an existing Atom entry from the collection into your message body.
   b. If you do not have any Atom entries in your collection yet, write the XML markup for your first Atom entry based on the examples in these topics.
   c. Check that the <entry> tag at the start of your Atom entry contains the namespace declaration xmlns="http://www.w3.org/2005/Atom", and add that namespace declaration if it is not present. In Atom documents sent from CICS, the elements typically do not include the atom: namespace prefix. The Atom namespace is defined as the default namespace in the element at the beginning of the Atom document, so you do not have to use the atom: prefix for the child elements. In the CICS documentation, the prefix is used in the element names for clarity. If you prefer to use the atom: namespace prefix in the elements in your request, change the namespace declaration to xmlns:atom="http://www.w3.org/2005/Atom".
   d. Add the element <?xml version="1.0" ?> before your Atom entry.
4. Fill in the appropriate elements of your Atom entry with the data that you require, and delete any elements that you do not use. You do not need to provide data for all the elements that are present in an Atom entry document sent out by CICS. When you post an entry to a collection managed by CICS, you can omit any elements that are not stored in the CICS resource, and also any elements that are stored in the CICS resource but are generated by CICS or the service routine. Typically, you can omit at least the following elements:
   - Elements containing time stamps, such as the <atom:updated> element (unless your service routine accepts user input for these)
   - The <atom:id> element
   - The <atom:contributor> element
   - The <atom:link> element
   - The <atom:rights> element

   If in doubt, include the element, and CICS or the service routine will ignore it if it is not wanted. RFC 5023 has the complete list of possible elements in Atom entries. For a summary of the elements that are required, allowed, or not used in Atom entries, see “Atom element reference for CICS” on page 316. For a description of the content of each element, see “<atom:entry> element” on page 311.
5. Send your request to the server.

Results

The server sends an HTTP response with the status code 200, indicating successful completion of the request, or a suitable error response. If the request is successful, the response contains a copy of the new Atom entry. Check the Atom entry in the response from the server against your original submission, to make sure that you are satisfied with the new entry.

If you receive an error response, read the message body for the response, and refer to the list of status codes that CICS provides to Web clients in Appendix C, "HTTP status code reference for CICS Web support," on page 381. If the error response indicates that there might be a problem with your request, such as the HTTP status code 400 (Bad Request or Invalid Request), check to see whether the CICS region has issued message DFHML0100. CICS uses the z/OS XML System Services parser to parse the XML markup of your Atom entry. If the parser finds a problem with the markup, CICS issues message DFHML0100 containing the return code and reason code from the parser. For an explanation of the return code and reason code, see the *z/OS XML System Services User's Guide and Reference*, which is available from http://www.ibm.com/servers/eserver/zseries/zos/xml/.

Example

This is a request to create a new Atom entry in a collection with the URL `http://www.example.com:80/web20/myfeed` :

```
POST /web20/myfeed HTTP/1.1
Host: www.example.com:80
Content-Type: application/atom+xml;type=entry
Content-Length: 763

<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom">
    <title>This is my posted entry</title>
    <summary>
    This is my new posted entry
    </summary>
    <category term="Test Feeds"/>
    <author>
     <name>Joe Bloggs</name>
     <email>JBloggs@example.com</email>
     <uri>http://www.example.com/JBloggs/</uri>
    </author>
    <content type="text/xml">
     <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
      <data_field>
       Here is content for my posted entry
     </data_field>
     </SAMPBIND >
    </content>
</entry>
```

This is the response that CICS sends:

```
HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Content-Length: 1029

<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app">
    <generator uri="http://www.ibm.com/cics/" version="6.6.0">
    CICS Transaction Server Version 4.1.0
    </generator>
    <link rel="self" href="http://www.example.com:80/web20/entry/10"/>
    <link rel="edit" href="http://www.example.com:80/web20/entry/10"/>
    <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:10</id>
    <title>This is my posted entry</title>
    <summary>
    This is my new posted entry
    </summary>
    <category term="Test Feeds"/>
    <rights>Copyright (c) 2009, Joe Bloggs</rights>
    <published>2009-04-23T15:00:44+00:00</published>
    <author>
```

```
      <name>Joe Bloggs</name>
      <email>JBloggs@example.com</email>
      <uri>http://www.example.com/JBloggs/</uri>
     </author>
     <app:edited>2009-04-23T15:00:44+00:00</app:edited>
     <updated>2009-04-23T15:00:44+00:00</updated>
     <content type="text/xml">
      <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
       <data_field>
        Here is content for my posted entry
       </data_field>
      </SAMPBIND >
     </content>
</entry>
```

Making PUT requests to Atom collections

A Web client can edit an existing Atom entry in a collection by making an HTTP PUT request to the URL of the Atom entry, as stated in the <atom:link rel="edit"> element of the entry.

About this task

You can only edit a single Atom entry at a time, using the URL of the individual Atom entry. You cannot edit multiple Atom entries in a single request. CICS rejects PUT requests made to the URL of a collection.

Procedure

1. Make an HTTP GET request to the URL of the Atom entry, as stated in the <atom:link rel="edit"> element of the entry, to retrieve a current copy of the entry and to obtain its entity tag from the ETag HTTP header on the response. "Making GET requests to Atom feeds or collections" on page 336 explains how to do this.
2. Begin your HTTP PUT request with a request line consisting of the PUT method, followed by the path component of the URL of the Atom entry, followed by `HTTP/1.1`, which is the HTTP version for the request. You may also include the scheme (HTTP or HTTPS) and the host name in the URL. For an explanation of request lines, see "HTTP requests" on page 12.
3. Write HTTP headers for the request as follows, each on a new line:
   - The Host header, giving the host name from the URL of the Atom entry, if you did not already include the host name in the request line.
   - The Authorization header, with any security information required to access your collection, such as a user ID and password for basic authentication.
   - The If-Match header, with the entity tag that you just obtained for the existing Atom entry. If you have to override this check, you may use an asterisk in place of an entity tag, which makes the server apply your edits to the Atom entry even if it has been changed by another agent since you obtained it. Use this option with caution.
   - The Content-Type header, with the value `application/atom+xml;type=entry`.
   - The Content-Length header, stating the length of your message body in bytes (octets). Fill in the value for this header when you have finished writing the message body.

   Put an additional carriage return line feed (CRLF) after the last HTTP header to give an empty line.

4. Copy the Atom entry into your message body, and add the element <?xml version="1.0" ?> before it. Check that the <entry> tag contains the namespace declaration xmlns="http://www.w3.org/2005/Atom".
5. Edit the content of the elements in the Atom entry that you want to change. CICS provides correct time stamps, and a service routine must do the same, so do not update the time stamps in the entry. For a description of the content of each element, see "<atom:entry> element" on page 311.
6. Send your request to the server.

Results

When your request is accepted, the server sends an HTTP response with the status code 200, indicating successful completion of the request, or a suitable error response. CICS returns a copy of the edited Atom entry as the body of the response, so that you can verify your changes if you want.

As for a POST request, if you receive an error response, read the message body for the response, and refer to the list of status codes that CICS provides to Web clients in Appendix C, "HTTP status code reference for CICS Web support," on page 381. For an error response that might indicate a problem with your request, check for message DFHML0100 from the CICS region, and consult the *z/OS XML System Services User's Guide and Reference* for an explanation of the return code and reason code.

Example

This is an HTTP PUT request to edit the Atom entry with the URL `http://www.example.com:80/web20/entry/10` :

```
PUT /web20/entry/10 HTTP/1.1
Host: www.example.com:80
Content-Type: application/atom+xml;type=entry
Content-Length: 1034
If-Match: c4826af12991fb102ef13099c927c2ac24e4caa2

<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app">
    <generator uri="http://www.ibm.com/cics/" version="6.6.0">
    CICS Transaction Server Version 4.1.0
    </generator>
    <link rel="self" href="http://www.example.com:80/web20/entry/10"/>
    <link rel="edit" href="http://www.example.com:80/web20/entry/10"/>
    <id>tag:http://www.example.com/web20/myfeed,2009-01-20:tsqueue:WB20TSQ:10</id>
    <title>This is my updated entry</title>
    <summary>
    This is my new updated entry
    </summary>
    <category term="Test Feeds"/>
    <rights>Copyright (c) 2009, Joe Bloggs</rights>
    <published>2009-04-23T15:00:44+00:00</published>
    <author>
     <name>Joe Bloggs</name>
     <email>JBloggs@example.com</email>
     <uri>http://www.example.com/JBloggs/</uri>
    </author>
    <app:edited>2009-04-23T15:00:44+00:00</app:edited>
    <updated>2009-04-23T15:00:44+00:00</updated>
    <content type="text/xml">
     <SAMPBIND xmlns="http://www.ibm.com/xmlns/prod/cics/atom/bindfile/sampbind">
      <data_field>
       Here is new content for my updated entry
```

```
      </data_field>
     </SAMPBIND >
    </content>
</entry>
```

Making DELETE requests to Atom collections

A Web client can delete an existing Atom entry from a collection by making an HTTP DELETE request to the URL of the Atom entry, as stated in the <atom:link rel="edit"> element of the entry.

About this task

You can only delete a single Atom entry from a collection at a time, using the URL for an individual Atom entry. You cannot delete a whole collection at once. CICS rejects DELETE requests made to the URL of the collection.

Procedure

1. Begin your HTTP DELETE request with a request line consisting of the method DELETE, followed by the path component of the URL of the Atom entry, followed by `HTTP/1.1`, which is the HTTP version for the request. You may also include the scheme (HTTP or HTTPS) and the host name in the URL. For an explanation of request lines, see "HTTP requests" on page 12.
2. Write HTTP headers for the request as follows, each on a new line:
   - The Host header, giving the host name from the URL of the Atom entry, if you did not already include the host name in the request line.
   - The Authorization header, with any security information required to access your collection, such as a user ID and password for basic authentication.

   Put an additional carriage return line feed (CRLF) after the last HTTP header to give an empty line. Do not include any message body.
3. Send your request to the server.

Results

When your request is accepted, the server sends an HTTP response with the status code 200, indicating successful completion of the request, or a suitable error response. The response does not contain a copy of the deleted entry.

Example

This is a request to delete the Atom entry with the URL `http://www.example.com:80/web20/entry/10` :

```
DELETE /web20/entry/10 HTTP/1.1
Host: www.example.com:80
```

How to handle Atom collection editing requests in your service routine

When you create a collection from an existing Atom feed that is provided by a service routine, update the program to take appropriate actions for POST, PUT, and DELETE requests, as well as GET requests, for the entries in the collection.

The interface between CICS and the service routine essentially works in the same way for a collection as it does for an ordinary Atom feed. CICS provides information about the client request to your program in the DFHATOMPARMS container. Your program must analyze the request and then respond by returning

the DFHATOMPARMS container, and other character containers as needed, to CICS. The sample service routine DFH$W2S1 demonstrates how to do this.

The extra task involved for a collection is that the program might have to action the client request by modifying records in the resource (a database, file, or other resource) that holds the data for the Atom entries for this feed. Instead of simply returning the data for a single Atom entry taken from the resource, the program might need to respond as follows:

- For a POST request, add a new record and populate it with the data for a new Atom entry that the client has supplied.
- For a PUT request, edit an existing record according to changes requested by the client.
- For a GET request, delete a record.

Depending on the request method, the response from the program might just be an HTTP status line to confirm the requested change, with no further data. The changes made by your service routine permanently modify the content of your resource, so you must make sure that appropriate security measures are in place.

For client requests using the POST and PUT methods, CICS supplies the request body in a container called DFHREQUEST. The client should send a request body containing a complete Atom entry document. Your program must parse the markup of the Atom entry document, match the elements in the Atom entry to the fields in the records in your resource, and use the content of the elements to create a new resource record or update an existing resource record. Your program must also provide content for some elements, such as date and time stamps.

The Atom Publishing Protocol (RFC 5023) allows servers much liberty to modify the metadata or content of entries that a client submits to them. Your service routine can usually ignore or override the content of elements in the client request if that is the best approach for your resource record, provided that the resulting entry meets the requirements for the Atom format. If you have a query about a specific element that you want to code your service routine to ignore or override, consult RFC 4287 and RFC 5023 to check that your proposed action is acceptable, and then go ahead.

When your service routine receives a client request relating to a collection, it must perform the tasks described in this section. The instructions assume that you have already written a service routine that responds to HTTP GET requests from Web clients for an Atom feed, as explained in “Writing a program to supply Atom entry data” on page 268, and demonstrated in the sample service routine DFH$W2S1.

When you have coded your service routine to perform these tasks, if resource and command security are active in your CICS region and in use for the collection, ensure that the user IDs for Web clients have the correct permissions to access the CICS resources and commands used by the service routine. For more information about security measures to protect Atom collections, see Chapter 26, “Security for Atom feeds,” on page 357.

Handling GET requests for Atom collections

When a Web client makes a GET request for an Atom collection, your service routine must respond with an entry or a series of entries from the collection.

Procedure

1. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHATOMPARMS container, which contains information about the request. The sample service routine DFH$W2S1 shows you how to do this. "DFHATOMPARMS container" on page 271 describes all the parameters that CICS passes in this container.
2. Check the value of the **ATMP_HTTPMETH** parameter to identify the request method. CICS returns an error or makes an appropriate response for methods other than GET, POST, PUT, and DELETE.
3. Use the values of the **ATMP_ATOMTYPE** and **ATMP_SELECTOR** parameters in the DFHATOMPARMS container to identify the Atom entry that your program must return to CICS in this instance.
   a. If **ATMP_SELECTOR** is null and **ATMP_ATOMTYPE** has the value "collection", the client did not specify a particular Atom entry. If possible, return the Atom entry in the collection that was edited most recently. If your resource cannot store this data, return the Atom entry that was added to the collection most recently.
   b. If **ATMP_SELECTOR** contains a selector value and **ATMP_ATOMTYPE** has the value "collection", return the entry identified by the selector value. This combination of values indicates that the client is requesting a second or subsequent entry from the collection.
   c. If **ATMP_SELECTOR** contains a selector value and **ATMP_ATOMTYPE** has the value "entry", return the entry identified by the selector value. This combination of values indicates that the client is requesting a single, known Atom entry from the collection.
4. When you have identified the Atom entry that is required for the GET request, return the entry from the collection as you would for an ordinary Atom feed, but take the following additional steps:
   a. Use the **ATMP_EDITED** parameter in the DFHATOMPARMS container to return the date and time at which this Atom entry was last edited. If you are using the resource handling parameters in the DFHATOMPARMS container, the **ATMP_EDITED_FLD** parameter has the name and length of the relevant field in the resource. If your resource does not store this data, return spaces, and CICS assumes the current date and time. "DFHATOMPARMS container" on page 271 documents the required format for this field.
   b. Use the **ATMP_ETAGVAL** parameter in the DFHATOMPARMS container to return the entity tag for the Atom entry, followed by its length. To create an entity tag, you can use the EXEC CICS BIF DIGEST command to calculate the SHA-1 digest of the resource record, or use another suitable method to produce an entity tag that complies with the HTTP/1.1 protocol requirements.
   c. If **ATMP_ATOMTYPE** has the value "collection", meaning that the client wants multiple entries, and your resource stores data about when the entries were last edited, return the **ATMP_NEXTSEL** parameter as the selector value for the next Atom entry in the collection. "Sequence for Atom entries" on page 252 explains the order in which you should return your Atom entries. Returning the entries in the order of editing helps your compliance with the Atom Publishing Protocol.
   d. If **ATMP_ATOMTYPE** has the value "collection", return the **ATMP_PREVSEL**, **ATMP_FIRSTSEL**, and **ATMP_LASTSEL** parameters as the selector values for the previous, first, and last Atom entries in the collection. CICS uses these values to construct <atom:link> elements containing links to other partial

lists of entries in the collection. For more information about these parameters, see "DFHATOMPARMS container" on page 271.

The steps to return an entry for an ordinary Atom feed are listed in "Writing a program to supply Atom entry data" on page 268.

Handling POST requests for Atom collections

When a Web client makes a POST request for an Atom collection, your service routine must create the new entry in the collection and return a copy of it to the client.

About this task

The Web client supplies a complete Atom entry document in the body of their HTTP POST request, and CICS passes this request body to the service routine in the DFHREQUEST container.

If the resource that holds the data for your Atom entries has an XML binding with an associated XMLTRANSFORM resource, you can use the CICS functions for transforming XML into application data to parse the markup of the Atom entry document that the Web client supplies. If an XMLTRANSFORM resource is available, CICS provides its name in the **ATMP_XMLTRANSFORM** parameter in the DFHATOMPARMS container. For more information about the TRANSFORM XMLTODATA command and instructions for using the data mapping functions, see the *CICS Application Programming Guide*.

If you do not have an XML binding for the resource that holds the data for your Atom entries, other facilities are available depending on the language of your service routine:

- If your program is written in C or Assembler, you can use the z/OS XML System Services parser to parse the markup of the entry.
- If your program is written in Enterprise COBOL, use the XML PARSE verb, as in the DFH0W2F1 COBOL sample service routine.
- If your program is written in Enterprise PL/I, use the PLISAXA library function.

Procedure

1. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHATOMPARMS container, which contains information about the request. The sample service routine DFH$W2S1 shows you how to do this. "DFHATOMPARMS container" on page 271 describes all the parameters that CICS passes in this container.
2. Check the value of the **ATMP_HTTPMETH** parameter to identify the request method. CICS returns an error or makes an appropriate response for methods other than GET, POST, PUT, and DELETE.
3. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHREQUEST container, which contains the new Atom entry. The sample service routine DFH$W2S1 shows you how to do this. CICS does not support other media types in Atom feeds, and rejects with a 415 status code any client request that is not described as an Atom entry, with the media type application/atom+xml, with or without the type=entry parameter.
4. Using the CICS functions for transforming XML into application data, or another suitable facility, parse the XML markup for the Atom entry to identify all the elements of the Atom entry for which CICS provides support and the resource that holds the data for your Atom entries has a suitable field to store

data. For a listing and description of all the elements of Atom entries for which CICS provides support, see "<atom:entry> element" on page 311. Ignore any elements that CICS does not support, or that your service routine does not recognize, or that you cannot store in a field in a record in the resource that holds the data for your Atom entries. Your Atom configuration file should already be set up to provide suitable defaults for any required elements for which your resource is not able to store data.

Tip: If you find that the Web client has submitted a request containing invalid XML markup or data, reject the request with the response code `atmp_resp_invalid_request`.

5. Create a new record in the resource that holds the data for your Atom entries, and populate its fields with the content of the elements that you have identified. Your service routine can include whatever level of error checking you feel is appropriate, depending on the nature of your client and the sensitivity of the resource that holds the data for your Atom entries. If you have to override any of the data provided by the client, you may do this, but check your action against RFC 4287 and RFC 5023 to verify that your override is valid. CICS ignores Atom IDs provided by Web clients, and your service routine should also do this. Substitute an Atom ID produced by completing the prototype Atom ID passed by CICS in the **ATMP_ATOMID** parameter in the DFHATOMPARMS container, or use another valid format. For more information about the format of Atom IDs, see "Atom IDs for Atom entries" on page 255
6. If your resource stores data for the time when the entry was last edited (<app:edited> element), when it was last updated (<atom:updated> element), or when it was first published (<atom:published> element), use the EXEC CICS ASKTIME and EXEC CICS FORMATTIME commands to produce a date and time stamp in the RFC 3339 format for the current date and time, and use this time stamp to populate those fields. If the client provides date and time stamps in the <atom:updated>, <atom:published>, or <app:edited> elements, you might want to generate a new date and time stamp to ensure accuracy and validity. For more information about handling date and time stamps, see "Date and time stamps for Atom entries" on page 254.
7. If the client request is successful, return the new entry as you would for an ordinary Atom feed, supplying the data that you just placed in the new resource record, but with the following exceptions to the normal process:
   a. Do not return an **ATMP_NEXTSEL** parameter.
   b. Use the EXEC CICS BIF DIGEST command to calculate the SHA-1 digest of the new resource record, or use another suitable method to produce an entity tag that complies with the HTTP/1.1 protocol requirements. Use the **ATMP_ETAGVAL** parameter in the DFHATOMPARMS container to return the result as an entity tag for the Atom entry, followed by its length.
   c. For the **ATMP_EDITED** parameter, you may either return the date and time stamp that you stored in your resource, or return spaces to allow CICS to provide the current date and time (provided that you did not specify an alternative default in your Atom configuration file).

   The sample service routine DFH$W2S1 shows you how to return an Atom entry to CICS. CICS creates the response with an HTTP status code of 201 and supplies a Location header to give the client the URI of the new Atom entry, and a matching Content-Location header so that the client knows the response is a complete representation of the Atom entry. The client can examine the entry in the body of the message to see any modifications that you made to the data supplied in its request.

8. If the client request is unsuccessful, use the **ATMP_RESPONSE** parameter in the DFHATOMPARMS container to return a suitable response code. When you return an error response, CICS produces a suitable default HTTP error response to send to the Web client. "DFHATOMPARMS container" on page 271 lists the response codes that you can use and the HTTP error response that CICS sends in each case.

Handling PUT requests for Atom collections

When a Web client makes a PUT request for an Atom collection, your service routine must update the entry and indicate whether or not the request was successful.

About this task

As with a POST request, the Web client supplies a complete Atom entry document in the body of their HTTP PUT request, and CICS passes this request body to the service routine in the DFHREQUEST container. If the resource that holds the data for your Atom entries has an XML binding with an associated XMLTRANSFORM resource, you can use the CICS functions for transforming XML into application data to parse the markup of the Atom entry document that the Web client supplies. If you do not have an XML binding for the resource that holds the data for your Atom entries, you can parse the markup of the Atom entry document using one of the other facilities listed in "Handling POST requests for Atom collections" on page 349.

Procedure

1. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHATOMPARMS container, which contains information about the request. The sample service routine DFH$W2S1 shows you how to do this. "DFHATOMPARMS container" on page 271 describes all the parameters that CICS passes in this container.
2. Check the value of the **ATMP_HTTPMETH** parameter to identify the request method. CICS returns an error or makes an appropriate response for methods other than GET, POST, PUT, and DELETE.
3. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHREQUEST container, which contains the updated Atom entry. The sample service routine DFH$W2S1 shows you how to do this.
4. Use the **ATMP_SELECTOR** parameter to select, from the resource that holds the data for your Atom entries, the record that contains the data for the Atom entry which the client wants to update.
5. Use the EXEC CICS BIF DIGEST command to calculate the SHA-1 digest of the current record in your resource that contains the data for the Atom entry, or calculate the entity tag using your service routine's alternative method, and compare this to the entity tag provided in **ATMP_ETAGVAL**. If the tags do not match, indicating that the data for the Atom entry has been changed by another agent since the Web client obtained its copy, reject the request with the response code `atmp_resp_etag_no_match`. If the entity tag is an asterisk, the Web client has chosen to override this process, and you should accept the request.
6. Using the CICS functions for transforming XML into application data, or another suitable facility, parse the XML markup for the Atom entry to identify all the elements of the Atom entry for which CICS provides support and the resource that holds the data for your Atom entries has a suitable field to store data. For a listing and description of all the elements of Atom entries for

which CICS provides support, see "<atom:entry> element" on page 311. Ignore any elements that CICS does not support, or that your service routine does not recognize, or that you cannot store in a field in a record in the resource that holds the data for your Atom entries. Your Atom configuration file should already be set up to provide suitable defaults for any required elements for which your resource is not able to store data.

Tip: If you find that the Web client has submitted a request containing invalid XML markup or data, reject the request with the response code `atmp_resp_invalid_request`.

7. Update the record in your resource that contains the data for the Atom entry, using the content of the elements that you have identified to make modifications to the fields in the record. In the body of a PUT request, the client is expected to provide the complete Atom entry including the changed and unchanged elements, so in theory you can update all the fields in the resource record without comparing them to see which ones have changed. However, depending on the nature of your client and the sensitivity of your resource, you might want to carry out error checking for any fields that have particular requirements for the format of their content, and check that the client has not changed the content of any fields where a change is not logical, such as the Atom ID or the time of first publication. If you have to override any of the data provided by the client, you may do this, but check your action against RFC 4287 and RFC 5023 to verify that your override is valid.
8. If your resource stores data for the time when the entry was last edited (<app:edited> element) and when it was last updated (<atom:updated> element), use the EXEC CICS ASKTIME and EXEC CICS FORMATTIME commands to produce a date and time stamp in the RFC 3339 format for the current date and time, and use this time stamp to populate those fields. If the client provides date and time stamps in the <atom:updated> or <app:edited> elements, ignore these, because they might simply be the previous date and time stamps returned unchanged. For more information about handling date and time stamps, see "Date and time stamps for Atom entries" on page 254.
9. If the client request is successful, return the updated entry as you would for an ordinary Atom feed, supplying the data from the updated resource record, but with the following exceptions to the normal process:
   a. Do not return an **ATMP_NEXTSEL** parameter.
   b. Use the EXEC CICS BIF DIGEST command to calculate the SHA-1 digest of the updated resource record, or use another suitable method to produce an entity tag that complies with the HTTP/1.1 protocol requirements. Use the **ATMP_ETAGVAL** parameter in the DFHATOMPARMS container to return the result as a new entity tag for the Atom entry, followed by its length.
   c. For the **ATMP_EDITED** parameter, you may either return the date and time stamp that you stored in your resource, or return spaces to allow CICS to provide the current date and time (provided that you did not specify an alternative default in your Atom configuration file).

   The sample service routine DFH$W2S1 shows you how to return an Atom entry to CICS. CICS creates the response with an HTTP status code of 200, indicating successful completion of the request. The client can examine the entry in the body of the message to see any modifications that you made to the data supplied in its request.
10. If the client request is unsuccessful, use the **ATMP_RESPONSE** parameter in the DFHATOMPARMS container to return a suitable response code. When you return an error response, CICS produces a suitable default HTTP error

response to send to the Web client. “DFHATOMPARMS container” on page 271 lists the response codes that you can use and the HTTP error response that CICS sends in each case.

Handling DELETE requests for Atom collections

When a Web client makes a DELETE request for an Atom collection, your service routine must delete the entry and indicate whether or not the request was successful.

Procedure

1. Use the EXEC CICS GET CONTAINER command to retrieve the data in the DFHATOMPARMS container, which contains information about the request. The sample service routine DFH$W2S1 shows you how to do this. “DFHATOMPARMS container” on page 271 describes all the parameters that CICS passes in this container.
2. Check the value of the **ATMP_HTTPMETH** parameter to identify the request method. CICS returns an error or makes an appropriate response for methods other than GET, POST, PUT, and DELETE.
3. Use the **ATMP_SELECTOR** parameter to select, from the resource that holds the data for your Atom entries, the record that contains the data for the Atom entry which the client wants to delete.
4. Delete the record in your resource that corresponds to the selector value in the **ATMP_SELECTOR** parameter, and use the **ATMP_RESPONSE** parameter in the DFHATOMPARMS container to return a response code of zero. CICS ignores the remaining parameters in the DFHATOMPARMS container, so you do not need to make any changes to these. CICS sends a response to the client with the status code 200, indicating successful completion of the request.
5. If you do not carry out the request, return an alternative response code in the **ATMP_RESPONSE** parameter. “DFHATOMPARMS container” on page 271 lists the response codes that you can use and the HTTP error response that CICS sends in each case.

DFH0W2F1 COBOL sample service routine for Atom feeds

The sample service routine DFH0W2F1 is a COBOL program that handles GET, POST, PUT, and DELETE requests for Atom entries that use data from the CICS sample file FILEA. You can use these interactions as a model for handling resources in your own service routine.

You can run DFH0W2F1 to handle Web client requests and supply data from the FILEA sample file for test or demonstration purposes. Before you can run DFH0W2F1, you must complete these tasks:

1. Create and install an appropriate RDO definition for DFH0W2F1, which must specify EXECKEY(CICS).
2. Create the FILEA VSAM file and install an appropriate RDO definition for it. An appropriate RDO definition is provided in the CICS-supplied sample RDO group DFH$FILA.
3. Ensure that the FILEA file is enabled and open.

CICS provides sample URIMAP and ATOMSERVICE resources in the DFH$WEB2 group that you can use to run DFH0W2F1. The resources are both named DFH$W2P1. You can set up these resources in the same way as you set up the sample Atom collection, following the instructions in the Web 2.0 scenario "Create an Atom feed for personnel locations" in the CICS TS for z/OS, Version 4.1

Information Center, which is available at `https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp`. The URL for Web client requests to access the FILEA sample as an Atom collection through DFH0W2F1 is

`http://host:port/atom/p/filea/feed`

where:

- *host* is the character host name, IPv4 address, or IPv6 address that is defined in the HOST attribute of the TCPIPSERVICE resource that you are using with this collection.
- *port* is the port number that is defined in the PORTNUMBER attribute of the TCPIPSERVICE resource that you are using with this collection.

The ATOMSERVICE resource DFH$W2P1 names the Atom configuration file dfh0w2f1.xml, which provides metadata and structure for the Atom feed document and Atom entries. dfh0w2f1.xml is in the /samples/web2.0/ subdirectory of the root directory for CICS files on z/OS UNIX, as specified by the CICS system initialization parameter USSHOME. The default value for USSHOME is /usr/lpp/cicsts/cicsts41. The ATOMSERVICE resource DFH$W2P1 does not name an XML binding for the FILEA file, because FILEA does not contain any fields that can be used as metadata for Atom entries, so DFH0W2F1 does not make use of the <cics:fieldnames> element in the Atom configuration file.

The DFH0W2F1 sample program performs the following tasks:

Working-Storage section

- Copies the copybook DFHW2CNO which has the constants for the response codes.
- Sets up local variables in the working storage.
- Includes the definition of the FILEA record layout, which is described in copybook DFH0CFIL which is shipped with CICS.
- Sets up a prototype that contains XML tagging and fields to hold the content of the Atom entry. A carriage return line feed (X'0D25') is used between each XML element. The service routine will produce the content of the Atom entry by reading a record in the FILEA file, inserting the data from the fields in the record into this prototype, and returning it in the DFHATOMCONTENT container.
- Sets up a prototype Atom entry title and summary that will be used in the DFHATOMTITLE and DFHATOMSUMMARY container. The records in the FILEA file do not contain titles and summaries, but the service routine will compose them using information from the file records. The title and summary that the service routine supplies replace the default title and summary that are specified in the Atom configuration file.
- Sets up content to return for an Atom entry to indicate an error if FILEA cannot be read. This is for demonstration purposes and is not appropriate for a production Atom feed.
- Sets up an XML stack that will be used when parsing Atom entries.
- Sets up storage to contain the entity tag (ETag) for the Atom entry.

Linkage section

- Copies the copybook DFHW2APO which describes the layout of the DFHATOMPARMS container.
- Defines a variable length string that can be used for any parameter in the DFHATOMPARMS container.

- Describes the storage that contains the request body from the Web client which was passed in the DFHREQUEST container.
- Defines storage to contain the selector values for the first, last, next, and previous records in the file, which CICS uses to create additional navigation links for collections.

Main program

- Uses the EXEC CICS GET CONTAINER command to obtain the parameters from the DFHATOMPARMS container.
- Obtains the selector value from the ATMP_SELECTOR parameter. The selector value is the key for a record in FILEA that contains the data for the requested Atom entry. In FILEA, the key for the records is a unique customer number. The key for FILEA must be exactly six digits, and DFH0W2F1 does not pad the key value to the correct length. If CICS does not supply a selector value, the service routine reads the first record in the file and copies its key to the ATMP_SELECTOR parameter storage. The service routine is returning the records in increasing order of their key, as CICS does for KSDS files.
- Saves the entity tag (Etag) value from the If-Match header, if an entity tag value was provided.
- Obtains the HTTP method for the request. The service routine uses the HTTP method to determine the subsequent action:
  - For a GET request, the service routine reads the record whose key is equal to the selector value, and returns its contents as an Atom entry using the RETURN-FILE-CONTENT procedure.
  - For a PUT request, the service routine reads the record whose key is equal to the selector value, using a read for update function. It calculates the ETag value for the record that was just read, and compares it with the one that was received from the If-Match header. If these are not equal, the file is unlocked from the read for update, and the service returns an error response. If the ETags match, the service routine uses the PARSE-REQUEST-BODY procedure to parse the updated Atom entry that was supplied by the Web client, and uses this data to update the record in FILEA. The file record is rewritten with a REWRITE function. The service routine then uses the RETURN-FILE-CONTENT procedure to return a copy of the updated Atom entry to the Web client.
  - For a POST request, the service routine uses the PARSE-REQUEST-BODY procedure to parse the updated Atom entry that was supplied by the Web client, and uses the data to write a new file record. The customer number that is provided by the Web client in the POST request becomes the key value for the new file record, and the service routine returns an error if a record with that number already exists. The service routine then uses the RETURN-FILE-CONTENT procedure to return a copy of the new Atom entry to the Web client.
  - For a DELETE request, the record whose key is equal to the selector value is deleted.

PARSE-REQUEST-BODY and XML-HANDLER procedures

In the PARSE-REQUEST-BODY procedure, the service routine uses an EXEC CICS GET CONTAINER command to get the content of the DFHREQUEST container, which is the Atom entry that was supplied by the Web client in the body of a POST or PUT request. It then parses the Atom entry using the XML PARSE function.

The XML-HANDLER procedure analyzes the elements of the supplied Atom entry, and uses the XML stack to track whether or not each XML element is within the <atom:content> element in the supplied Atom entry ("IN-CONTENT" flag), and record the nesting level of each element. When an element is found that is "IN-CONTENT" and has a recognized name, the UPDATE-RECORD-FIELD procedure saves its data in the FILEA record. The service routine ignores metadata elements in the Atom entry that are outside the <atom:content> element, and also any elements within the <atom:content> element whose name it does not recognize, because it cannot store the data from these elements in the FILEA records.

The COBOL XML parser does not attempt rigorous analysis of XML namespace prefixes. If you are using this parser, the <atom:content> element in the supplied Atom entry must include the prefix "atom", and any XML elements within the content must not have namespace prefixes.

RETURN-FILE-CONTENT procedure

For GET, POST, and PUT requests, the RETURN-FILE-CONTENT procedure returns the data from a record in the FILEA file as the content of an Atom entry, and creates and returns a title and summary for the Atom entry. The procedure performs the following steps:

- Places the data from the fields of the FILEA record into the prototype Atom entry content that was set up in the Working-Storage section. The resulting XML structure is placed into the DFHATOMCONTENT container with an EXEC CICS PUT CONTAINER command.
- Completes the prototype Atom entry title and saves it in the DFHATOMTITLE container. A flag indicating this is set for the ATMP-OPTIONS-OUT parameter.
- Creates a summary for the Atom entry and saves it in the DFHATOMSUMMARY container. A flag indicating this is set for the ATMP-OPTIONS-OUT parameter.
- Uses the EXEC CICS BIF DIGEST command to produce an ETag for the FILEA record, and returns this value in the ATMP-ETAGVAL parameter in the DFHATOMPARMS container.
- Calls

ADD-NAVIGATION-LINKS procedure

The ADD-NAVIGATION-LINKS procedure returns the selector values (that is, the key values) of the first, last, next, and previous records in the FILEA file. The selector values are saved in the temporary work area (TWA). CICS will use the selector values for creating additional navigation for Atom collection documents.

Chapter 26. Security for Atom feeds

CICS Web support provides a suitable security protocol and authentication method to control Web client access to Atom collections and if required, to Atom feeds. You can use CICS resource and command security to protect the resources that you use to deliver the Atom feed or collection.

RFC 5023 recommends that you use authentication to protect Atom collections. When you make Atom feed data available as an editable collection, a Web client can insert new entries, modify existing entries, or delete entries. You must therefore ensure that you verify the identity of Web clients and permit only trusted clients to have access to the collection, especially if you have included business data in your collection. Ordinary Atom feeds, which Web clients cannot edit, are typically made available to any subscribers without security restrictions, although you might need to restrict access to Atom feeds if they include confidential business data or are intended only for certain users.

RFCs 4287 and 5023 discuss the use of digital signatures and encryption for Atom documents. CICS does not provide support for digital signatures and encryption of Atom documents, but, in compliance with RFC 4287, CICS does not reject an Atom document that contains a signature.

CICS Web support has the following security functions that you can use to protect Atom feeds or collections from unauthorized access or updates:

SSL or TLS security protocol
: RFC 5023 recommends the use of the Transport Layer Security (TLS) 1.0 as a minimum level of security protocol for collections. CICS supports the Secure Sockets Layer (SSL) 3.0 protocol and the Transport Layer Security (TLS) 1.0 protocol. The *CICS RACF Security Guide* explains the facilities that SSL and TLS provide.

HTTP basic authentication
: RFC 5023 recommends the use of HTTP basic authentication as a minimum level of authentication for collections. "HTTP basic authentication" on page 20 explains this mechanism.

Client certificate authentication
: Client certificate authentication is a more secure method of authenticating a client, using a client certificate which is issued by a trusted third party (or Certificate Authority), and sent using SSL encryption. The *CICS RACF Security Guide* explains how this works.

When you set up these functions in CICS Web support, you can apply them to an Atom feed or collection using attributes of the TCPIPSERVICE definition for the port where CICS receives Web client requests for the Atom feed or collection. For information on setting up SSL support for CICS Web support, see the *CICS RACF Security Guide*. For information about setting up basic authentication or client certificate authentication for CICS Web support, see "CICS as an HTTP server: authentication and identification" on page 173.

Resource and command security for Atom feeds

CICS resource security and command security prevents unauthorized access to the CICS resources that provide the data for Atom feeds, including files, transient data queues, and ATOMSERVICE resource definitions, and also prevents the execution of unauthorized system commands through service routines that are used to deliver Atom feeds. You can permit some clients read-only access to a feed or collection (HTTP GET requests only), and give other clients permission to update the resources so that they can make all types of HTTP request.

For resource security, you define ATOMSERVICE resource definitions to RACF in the RCICSRES class or the WCICSRES grouping class, or your equivalent classes that you have specified in the XRES system initialization parameter. Command security for ATOMSERVICE resources uses the ATOMSERVICE resource in the CCICSCMD class or the VCICSCMD grouping class. You can also apply resource security to other types of resource and command used in delivering the Atom feed or collection, using the classes listed for those resources in the *CICS RACF Security Guide*.

If resource and command security are active in your CICS region, to use these functions for an Atom feed or collection, define its alias transaction with RESSEC(YES) and CMDSEC(YES). You name the alias transaction for an Atom feed or collection in the TRANSACTION attribute of the URIMAP definition for the feed or collection. The default alias transaction for Atom feeds is CW2A, which is already defined with RESSEC(YES) and CMDSEC(YES). For more information about alias transactions for Atom feeds, see "Creating an alias transaction for an Atom feed" on page 294.

Resource and command security for Atom feeds works in the same way as for an application-generated response from CICS Web support. When you specify resource and command security for an alias transaction, you must give all the possible user IDs for your alias transaction permissions to use the resources accessed and commands used by the transaction, including the ATOMSERVICE resource definition. User IDs to which you give READ access can obtain entries from the feed or collection using HTTP GET requests, but they are not authorized to make other types of request. User IDs to which you give UPDATE access can make POST, PUT, and DELETE requests to amend the data in the resources. If you have obtained an authenticated user ID for a Web client, this user ID is applied to the alias transaction, or you can supply your own standard user ID as an override. For an explanation of how the user IDs for Web clients are used in security checking, and instructions to set up resource and command security for a Web alias transaction, see "Resource and transaction security for application-generated responses" on page 180.

If a Web client is not authorized to access the resources used by the alias transaction, CICS issues the security violation message DFHXS1111 and returns an HTTP error response with the status code 403 (Forbidden) to the client.

Part 4. The CICS business logic interface

Information about the CICS business logic interface.

Chapter 27. Introduction to the CICS business logic interface

The CICS business logic interface makes it possible to link to a Web-aware business application, rather than invoking it through the CICS HTTP listener.

For example, a Web server running on z/OS can use the external CICS interface (EXCI) to link to an application using the CICS business logic interface. In this way, a Web client can communicate with a CICS application through an intermediate Web server, rather than making a direct connection to CICS.

Appendix G, "Reference information for DFHWBBLI, CICS business logic interface," on page 413 has reference information for the interface.

How the CICS business logic interface is used

You can call the CICS business logic interface in any environment where you can link to a CICS application program.

For example:

- You can issue an **EXEC CICS LINK** command from a CICS application program.
- You can use the external CICS interface (EXCI).
- You can use the external call interface (ECI) from a client.
- You can use CICS ONC RPC support from an ONC RPC client.

The CICS business logic interface is used by:

- The CICS Web server plug-in. The plug-in uses the external CICS interface to invoke the CICS business logic interface.

Processing examples

Examples of how the CICS business logic interface processes a request from an MVS application that uses either the EXCI, or the ECI.

Figure 19 shows how the CICS business logic interface processes a request from an MVS application that uses the EXCI.

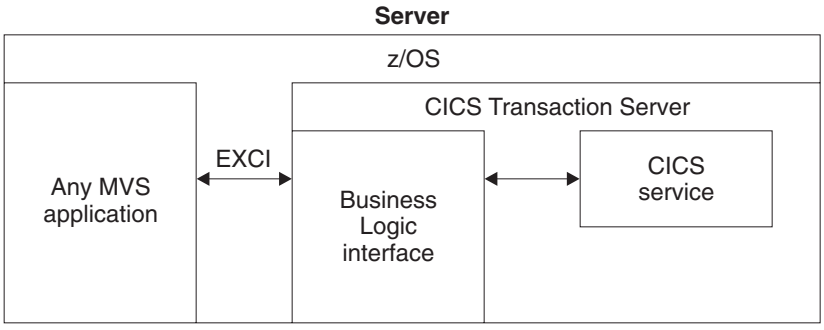


Figure 19. Processing a request from the EXCI

1. The MVS application constructs a COMMAREA that contains parameters for the CICS business logic interface.

2. The MVS application uses the EXCI to call the CICS business logic interface.
3. The CICS business logic interface calls the requested service, and returns any output in the COMMAREA.

Figure 20 shows how the CICS business logic interface processes a request from a CICS client that uses the ECI.

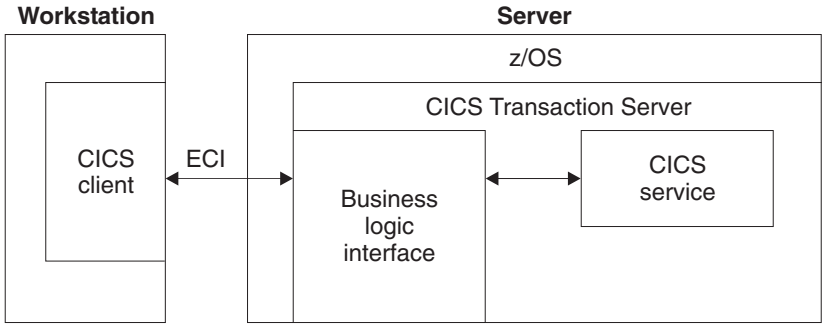


Figure 20. Processing a request from the ECI

1. The client, running in a workstation environment, constructs a COMMAREA that contains parameters for the CICS business logic interface.
2. The client uses the ECI to call the CICS business logic interface.
3. The CICS business logic interface calls the requested service, and returns any output in the COMMAREA.

The ECI operates with either the SNA protocol or with TCP62, which allows a SNA connection over TCP/IP (see the *CICS Family: Client/Server Programming* for further information).

Control flow in request processing

To make decisions about the facilities you will use, and how you will customize them, you need to understand how the components of the CICS business logic interface interact.

Using the CICS business logic interface to call a program

Figure 21 on page 363 shows the control flow through the CICS business logic interface to a program. The CICS business logic interface is accessed by a LINK command to PROGRAM DFHWBBLI.

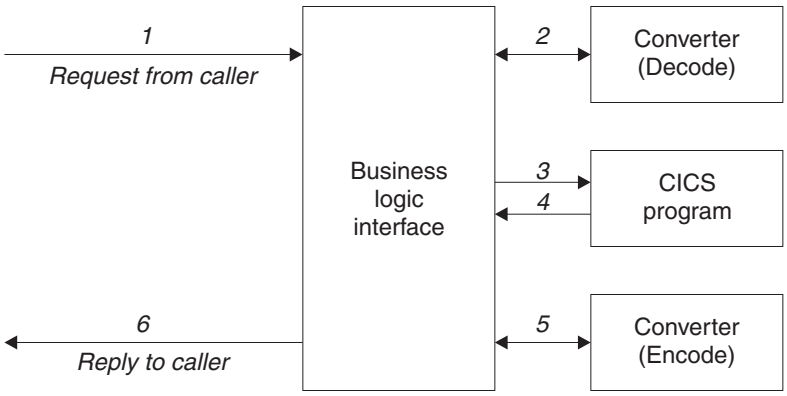


Figure 21. Calling a program with the CICS business logic interface—control flow

1. A request arrives for the CICS business logic interface.
2. If the caller requests a converter, the CICS business logic interface calls it, requesting the **Decode** function. **Decode** sets up the COMMAREA for the CICS application program.
3. The CICS business logic interface calls the CICS application program that the caller specified. The COMMAREA passed to the application program is the one set up by Decode. If the caller of the CICS business logic interface indicates that a converter is not required, the first 32K bytes of the request is passed to the CICS application program in its COMMAREA.
4. The CICS application program processes the request, and returns output in the COMMAREA.
5. If the caller requested a converter, the CICS business logic interface calls the **Encode** function of the converter, which uses the COMMAREA to prepare the response. If no converter program was called, the CICS business logic interface assumes that the CICS application program has put the desired response in the COMMAREA.
6. The CICS business logic interface sends a reply back to the caller.

Using the CICS business logic interface to run a terminal-oriented transaction

Figure 22 on page 364 shows the control flow through the CICS business logic interface for a request for a terminal-oriented transaction. Note that the business logic interface is running under a CICS mirror transaction, not a Web CICS transaction. The first part of the processing is the same as for calling a program, but if you want to run a transaction, you must specify DFHWBTTA as the CICS application program to be called, in **wbbl_server_program_name**.

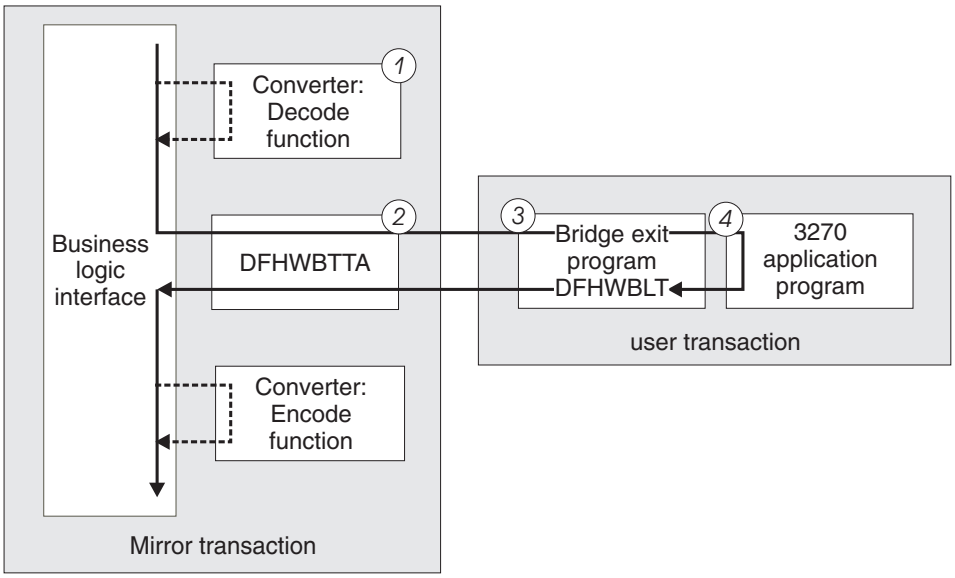


Figure 22. Running a transaction with the CICS business logic interface—control flow

1. If the caller requests a converter, the CICS business logic interface calls it, requesting the **Decode** function. **Decode** sets up the COMMAREA for DFHWBTTA.
2. The CICS business logic interface calls DFHWBTTA. The COMMAREA passed to DFHWBTTA is the one set up by **Decode**. If no converter program was called, the COMMAREA contains the entire request.
3. DFHWBTTA extracts the transaction ID for the terminal-oriented transaction from the HTTP request, and starts a transaction that runs the CICS Web bridge exit.
4. When the program attempts to write to its principal facility, the data is intercepted by the CICS Web bridge exit. The exit constructs the HTML response which is returned to the CICS business logic interface. If the caller requested a converter, the CICS business logic interface calls the **Encode** function of the converter, which uses the COMMAREA to prepare the response. If no converter program was called, the CICS business logic interface assumes that the COMMAREA contains the desired response.

Data flow in request processing

To make decisions about the facilities you will use, and how you will customize them, you need to understand how data is passed in the CICS business logic interface.

Converter programs and the CICS business logic interface

You can have many converter programs in a CICS system to support the operation of the CICS business logic interface.

The place of converters in the CICS business logic interface is illustrated in Figure 21 on page 363 and Figure 22. Each converter must provide two functions:

- **Decode** is used before the CICS application program is called. It can:
  - Use the data from the incoming request to build the COMMAREA in the format expected by the application program.

  - Supply the lengths of the input and output data in the application program's COMMAREA.
  - Perform administrative tasks related to the request.
- **Encode** is used after the CICS application program has been called. It can:
  - Use the data from the application program to build the response.
  - Perform administrative tasks related to the response.

Notes:

- If DECODE_DATA_PTR or ENCODE_DATA_PTR has been altered to address another storage location, it is the converter program's responsibility to freemain the original storage.
- It is the responsibility of the caller of the CICS business logic interface to free the buffer addressed by ENCODE_DATA_PTR (that is, the address returned in field WBBL_OUTDATA_PTR minus 4).
- If the converter abends, CICS will attempt to free the storage addressed by DECODE_DATA_PTR and ENCODE_DATA_PTR. Therefore, you should ensure that these pointers never contain the address of storage that has already been freed.

Using the CICS business logic interface to call a program

Data flows through the CICS business logic interface to a program, and back to the requester.

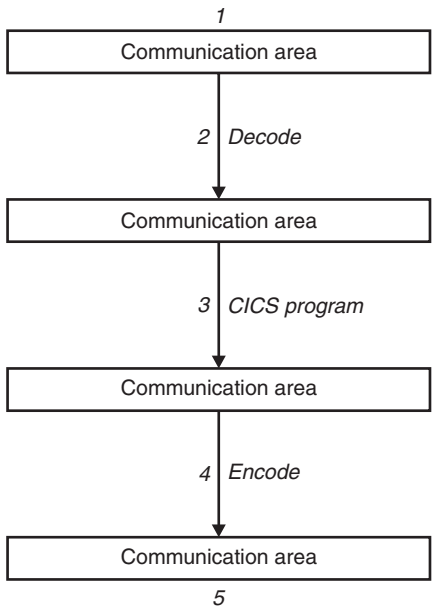


Figure 23. Calling a program with the CICS business logic interface—data flow

1. The caller of the CICS business logic interface provides a COMMAREA that contains the request to be processed. The contents of the COMMAREA must be in a code page acceptable to the subsequent processes. Usually this means that they must be in EBCDIC.
2. If the caller requests a converter, the **Decode** function of the converter constructs the COMMAREA for the CICS application program.
3. The CICS application program updates the COMMAREA.

4. If the caller requests a converter, the **Encode** function of the converter constructs the COMMAREA that is to be returned to the caller.
5. The CICS business logic interface returns to its caller, which can now use the contents of the COMMAREA.

Request for a terminal-oriented transaction

Figure 24 shows the data flow for a request that starts a terminal-oriented transaction.

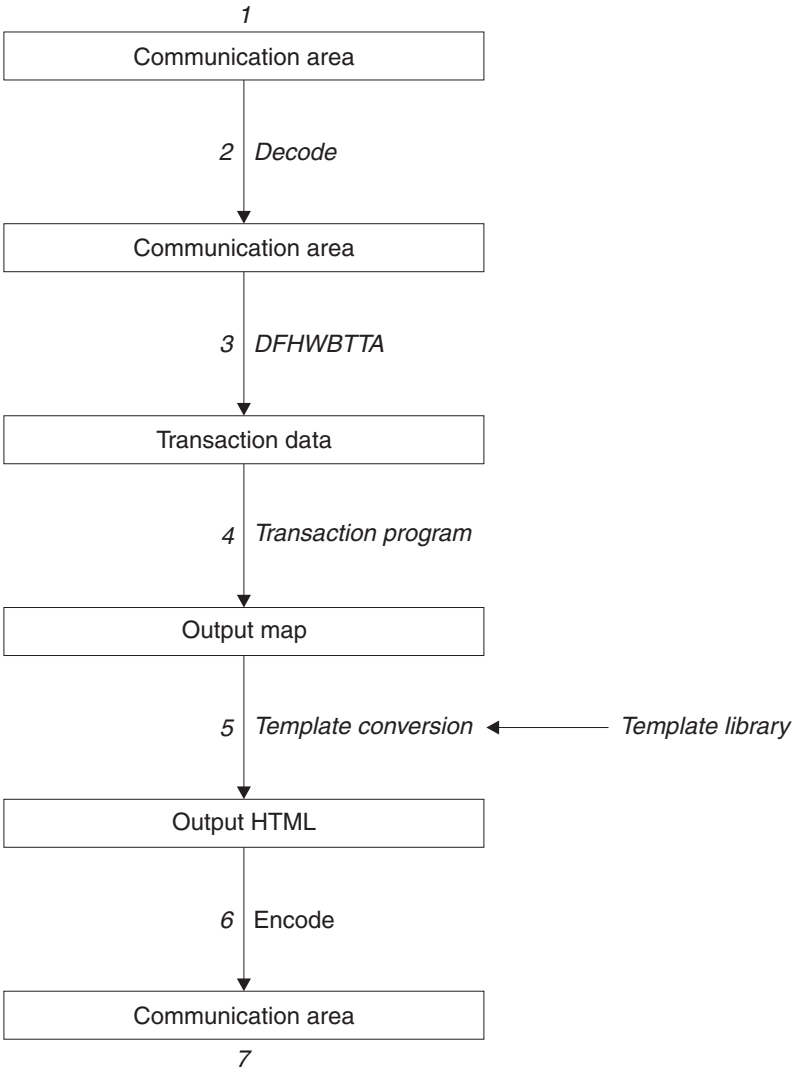


Figure 24. Starting a terminal-oriented transaction—data flow

This figure shows the data flow through the CICS business logic interface for a 3270 BMS application.

1. The caller of the CICS business logic interface provides a COMMAREA that contains the request to be processed. The contents of the COMMAREA must be in a code page acceptable to the subsequent processes, and DFHWBTTA requires EBCDIC.
2. You can use the **Decode** function of the converter to modify the request if required.

3. As this is the first transaction of a conversation or pseudoconversation, the request includes the transaction ID, and perhaps data to be made available to the transaction program. DFHWBTTA extracts the data so that it can be made available to the transaction program in a RECEIVE command.
4. The transaction program uses a RECEIVE command to receive the data. It then constructs an output map, and uses a SEND MAP command to send it to the requester.
5. The map and its data contents are converted into HTML. This conversion uses templates defined in DOCTEMPLATE definitions.
6. You can use the **Encode** function of the converter to modify the response if required.
7. The CICS business logic interface returns to its caller, which can now use the contents of the COMMAREA.

Figure 25 on page 368 shows the data flow for a request that continues a terminal-oriented transaction.

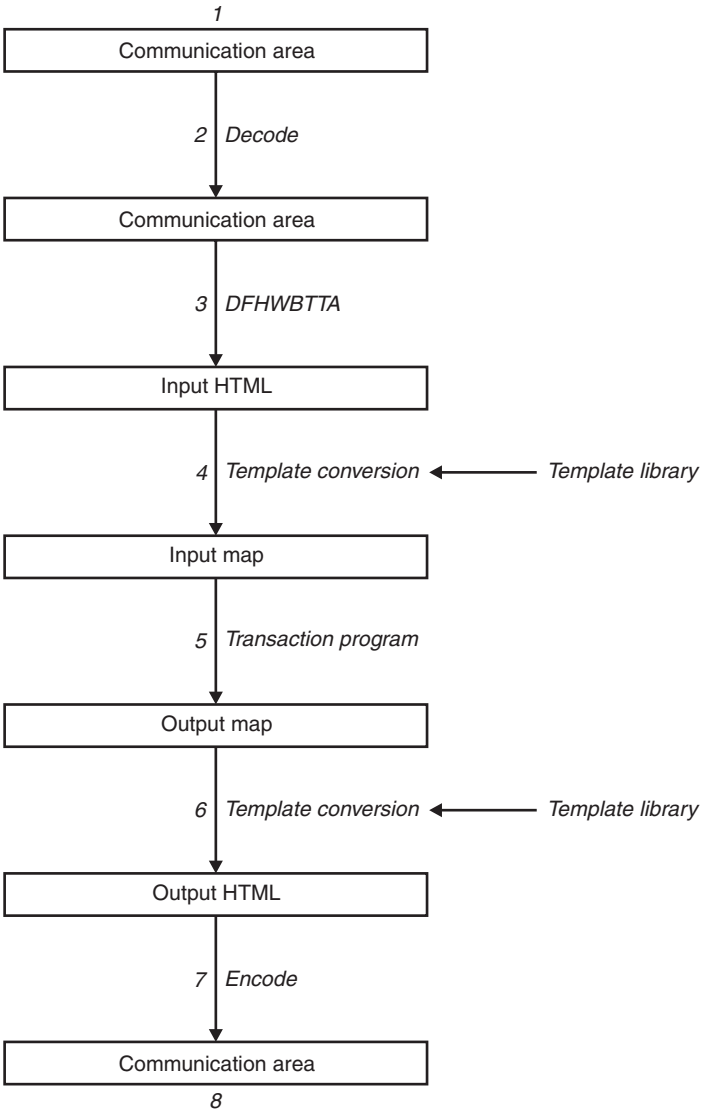


Figure 25. Continuing a terminal-oriented transaction—data flow

This figure shows the data flow when the CICS business logic interface processes the request.

1. The caller of the CICS business logic interface provides a COMMAREA that contains the request to be processed. The contents of the COMMAREA must be in a code page acceptable to the subsequent processes. Usually this means that they must be in EBCDIC.
2. The **Decode** function of the converter constructs the COMMAREA for DFHWBTTA.
3. As this is not the first transaction of a conversation or pseudoconversation, the request includes HTML corresponding to the map that the transaction program is expecting to receive. DFHWBTTA extracts the forms data to make it available to the transaction program in a RECEIVE MAP command.
4. The incoming forms input data is converted into a BMS map. This conversion uses templates from DOCTEMPLATE definitions.

5. The transaction program uses a RECEIVE MAP command to receive the data. It then constructs an output map, and uses a SEND MAP command to send it to the requester.
6. The map and its data contents are converted into HTML. This conversion uses templates from DOCTEMPLATE definitions.
7. The **Encode** function of the converter uses the HTML output from the conversion process to construct the COMMAREA to be returned to the caller.
8. The CICS business logic interface returns to its caller, which can now use the contents of the COMMAREA.

Offset mode and pointer mode

The CICS business logic interface can be called in two modes:

Offset mode

In offset mode, there is a single storage area (Storage area 1 in Figure 26) which contains DFHWBBLI's COMMAREA and the CICS application program's area. Field *wbbl_indata_offset* in DFHWBBLI's COMMAREA contains the offset of the application program's COMMAREA from the start of the storage area. The maximum size of the storage area is 32k bytes.

In offset mode, your converter program must not change the values of DECODE_DATA_PTR or ENCODE_DATA_PTR.

Pointer mode

In pointer mode, there are two independent storage areas: One (Storage area 1 in Figure 26) contains DFHWBBLI's COMMAREA and the other (Storage area 2) contains the CICS application program's area. Field *wbbl_indata_ptr* in DFHWBBLI's COMMAREA contains the address of the application program's COMMAREA.

In pointer mode, your converter program can change the values of DECODE_DATA_PTR or ENCODE_DATA_PTR.

The two modes are illustrated in Figure 26.

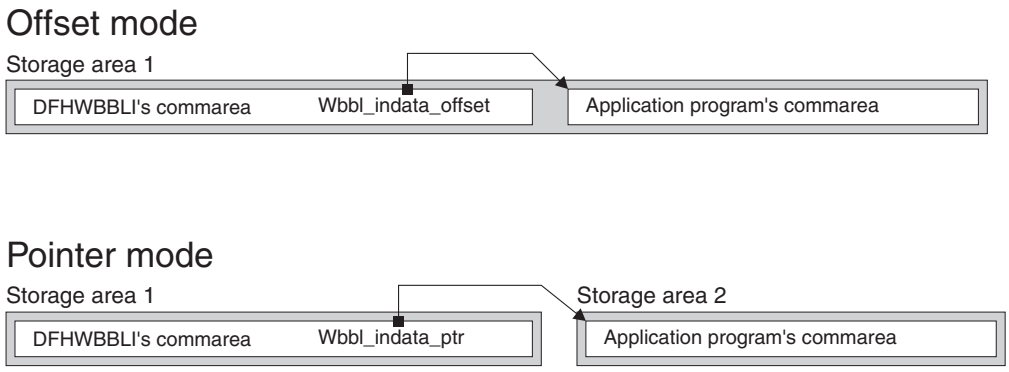


Figure 26. Offset mode and pointer mode in the CICS business logic interface

When you call the CICS business logic interface, you must specify the mode:

- Set wbbl_mode to "D" to indicate offset mode and that the body of the HTTP request (referenced by wbbl_user_data_offset) is in ASCII. This is required if the server program uses any of the FORMFIELD API commands.
- Set wbbl_mode to "P" to indicate pointer mode.

In your converter program, you can test `decode_volatile` or `encode_volatile` to determine the mode:

- 0 indicates offset mode
- 1 indicates pointer mode

All requests from any of the following sources use offset mode when calling the CICS business logic interface:

- Web clients using the IBM HTTP Server.
- Java applications using the local gateway function.
- DCE RPC clients.
- Web clients using the CICS Transaction Gateway.

Code page conversion and the CICS business logic interface

The CICS business logic interface does not perform code page conversion; the data that you pass to the business application, and the data that is returned is in the code page used by the application programming interface.

However, the **EXEC CICS WEB** application programming commands allow you to specify the client code page, and the data is converted in the application programming interface itself. Therefore, when you use these commands, code page conversion *is* performed between the application program and the CICS business logic interface. The data passed across the interface is in the code page specified in the CHARACTERSET option of the **EXEC CICS WEB** commands (or its synonym CLNTCODEPAGE).

Configuring the CICS business logic interface

About this task

Procedure

1. You must set the **WEBDELAY** system initialization parameter, as described in "Specifying system initialization parameters for CICS Web support" on page 54.
2. If you are not using autoinstall for programs, you must define all the user-replaceable programs (converters) that the callers of the CICS business logic interface use. If you are using autoinstall for programs, you do not need to define the converters. All the converters must be local to the system in which the CICS business logic interface is operating.

Part 5. Appendixes

Appendix A. HTML coded character sets

This reference lists the supported IANA-registered character set names (specified as `charset=` values in HTTP headers), and the IBM CCSID equivalents.

All of these values are valid for code page conversion options on the following commands:

- WEB RECEIVE (Client)
- WEB RECEIVE (Server)
- WEB SEND (Client)
- WEB SEND (Server)
- WEB CONVERSE
- DOCUMENT RETRIEVE
- WEB READ FORMFIELD
- WEB STARTBROWSE FORMFIELD

Table 14. Coded character sets

Language	Coded character set	IANA charset	IBM CCSID
Albanian	ISO/IEC 8859-1	iso-8859-1	819
Arabic	ISO/IEC 8859-6	iso-8859-6	1089
Bulgarian	Windows 1251	windows-1251	1251
Byelorussian	Windows 1251	windows-1251	1251
Catalan	ISO/IEC 8859-1	iso-8859-1	819
Chinese (simplified)	GB	gb2312	1381 or 5477
Chinese (traditional)	Big 5	big5	950
Croatian	ISO/IEC 8859-2	iso-8859-2	912
Czech	ISO/IEC 8859-2	iso-8859-2	912
Danish	ISO/IEC 8859-1	iso-8859-1	819
Dutch	ISO/IEC 8859-1	iso-8859-1	819
English	ISO/IEC 8859-1	iso-8859-1	819
Estonian	ISO/IEC 8859-1	iso-8859-1	819
Finnish	ISO/IEC 8859-1	iso-8859-1	819
French	ISO/IEC 8859-1	iso-8859-1	819
German	ISO/IEC 8859-1	iso-8859-1	819
Greek	ISO/IEC 8859-7	iso-8859-7	813
Hebrew	ISO/IEC 8859-8	iso-8859-8	916
Hungarian	ISO/IEC 8859-2	iso-8859-2	912
Italian	ISO/IEC 8859-1	iso-8859-1	819
Japanese	Shift JIS EUC Japanese	x-sjis or shift-jis euc-jp	943 (932, a subset of 943, is also valid) 5050 (EUC)
Korean	EUC Korean	euc-kr	970 (for AIX® or Unix)

Table 14. Coded character sets (continued)

Language	Coded character set	IANA charset	IBM CCSID
Latvian	Windows 1257	windows-1257	1257
Lithuanian	Windows 1257	windows-1257	1257
Macedonian	Windows 1257	windows-1257	1251
Norwegian	ISO/IEC 8859-1	iso-8859-1	819
Polish	ISO/IEC 8859-2	iso-8859-2	912
Portuguese	ISO/IEC 8859-1	iso-8859-1	819
Romanian	ISO/IEC 8859-2	iso-8859-2	912
Russian	Windows 1251	windows-1251	1251
Serbian (Cyrillic)	Windows 1251	windows-1251	1251
Serbian (Latin 2)	Windows 1250	windows-1250	1250
Slovakian	ISO/IEC 8859-2	iso-8859-2	912
Slovenian	ISO/IEC 8859-2	iso-8859-2	912
Spanish	ISO/IEC 8859-1	iso-8859-1	819
Spanish	ISO/IEC 8859-15	iso-8859-15	923
Swedish	ISO/IEC 8859-1	iso-8859-1	819
Turkish	ISO/IEC 8859-9	iso-8859-9	920
Ukrainian	Windows 1251	windows-1251	1251
Unicode	UCS-2	iso-10646-ucs-2	1200 (growing) or 13488 (fixed)
Unicode	UTF-16	utf-16	1200
Unicode	UTF-16 big-endian	utf-16be	1201
Unicode	UTF-16 little-endian	utf-16le	1202
Unicode	UTF-8	utf-8	1208

Appendix B. HTTP header reference for CICS Web support

In CICS Web support, CICS automatically provides some HTTP headers on outbound messages and you can also add your own headers. When messages are sent to CICS, CICS takes action in response to some HTTP headers, and a user application program can take action in response to others. This reference information describes how CICS Web support handles HTTP headers.

The standard HTTP headers are described in the HTTP/1.1 specification (RFC 2616) and the HTTP/1.0 specification (RFC 1945). There are many possible HTTP headers, including extension headers that are not part of the HTTP protocol specifications. For fuller listings, consult the HTTP specification to which you are working. "The HTTP protocol" on page 12 has more information about the HTTP specifications.

This topic explains the general use of HTTP headers in CICS Web support, and the actions that CICS Web support takes for specific headers. Check the HTTP specification to which you are working for detailed guidance and requirements about how you should use HTTP headers, such as the correct format for header values, and the contexts in which each header should be used.

HTTP headers on messages received by CICS

- When an HTTP request or response is received by CICS, some of the HTTP headers are used to determine actions that CICS Web support takes. Table 15 on page 377 shows the actions taken by CICS for headers on an HTTP request. Table 16 on page 378 shows the actions taken by CICS for headers on an HTTP response. Other headers are not used by CICS, and it is up to the user application to take appropriate action in response to these.
- All headers received for a message, whether or not they have been used by CICS, are made available to a user application for inspection using the WEB READ HTTPHEADER command and the HTTP header browsing commands. CICS does not alert the user application to the presence of any particular header on a message. Ignore any headers that the application does not need or understand.
- CICS already deals with the MUST level requirements in the HTTP/1.1 specification relating to actions that the server or client must perform on receiving a message. Because of this, you may receive and use a request or response without examining the headers. However, you will probably need to examine the headers for information relating to actions that you take in future communications with the Web client or server.
- HTTP headers consist of a header name and header value, separated by a colon. The HTTP/1.1 specification states that a single space is preferred between the colon and the header value, and that this common form should be followed. In the HTTP/1.0 specification, this single space was a requirement, but the HTTP/1.1 specification permits applications to use more spaces or no spaces. To preserve backwards compatibility, CICS requires the common form of a single space in some headers where action is taken by CICS during message processing (such as the Content-Length header). If you are designing an application that sends HTTP requests to CICS, ensure that this common form is followed for all HTTP headers, as recommended in the HTTP/1.1 specification. The EXEC CICS

WEB WRITE HTTPHEADER command produces headers with the appropriate format, and any headers written automatically by CICS are in the appropriate format.

HTTP headers on messages sent out from CICS

- On an HTTP request or response that is sent out from CICS with HTTP/1.1 as its version, CICS automatically supplies key headers that should normally be written for a basic message to be compliant with the HTTP/1.1 specification. On an HTTP response with HTTP/1.0 as its version, CICS automatically supplies a smaller number of headers. Some of these headers are generated by CICS for every message, and some are produced because of options that you specify on the WEB SEND command in a user application program. Table 17 on page 379 and Table 18 on page 380 list the headers that are written for each HTTP version, and the source of the header.

  If the user application program writes a header that CICS also generates, CICS handles this depending on the situation:

  – For CICS as an HTTP server, if the header is appropriate for a response, CICS does not overwrite it, but allows the application's version to be used.
  – For CICS as an HTTP client, if the header is appropriate for a request, CICS does not allow the application to write it, and returns an error response to the WEB WRITE HTTPHEADER command. The exceptions are the TE header and the Content-Type header. Application programs can add further instances of the TE header. They can also supply the Content-Type header, if the required header needs to contain spaces or more than 56 characters, and so cannot be specified on the MEDIATYPE option of the WEB SEND command.
  – If the header is not normally appropriate for the type of message (request or response), CICS allows it, as is the case for all user-defined headers. This situation should not occur if your message is compliant with the HTTP specification to which you are working.
- A user application program can add further HTTP headers to a request or response using the **WEB WRITE HTTPHEADER** command. CICS tolerates and passes on any additional HTTP headers. Note that for CICS as an HTTP server, if you are providing a static response with a CICS document template or HFS file, headers cannot be added to the response beyond those that are automatically supplied by CICS.
- CICS does not check the name or value of user-written headers. You should ensure that your application program is providing correct, and correctly formatted, information in a way that meets the HTTP specification to which you are working. Be particularly careful to check the HTTP specification for applicable requirements if your application is performing complex actions. There are likely to be important (MUST or SHOULD level) requirements to provide certain headers to describe these actions. For example, special HTTP headers are required if you are:

  – Responding to, or making, conditional requests using the modification date of the document or an entity tag.
  – Varying the content of a response according to the client capability or national language requirements of the Web client.
  – Providing a response, or making a request, that involves a range of a document rather than the full document.
  – Providing cache control information for a response.

  The use of certain status codes on your response might also require particular HTTP headers. For example, if you use the status code 405 (Method not allowed), you must use the Allow header to state the methods which are

allowed. Appendix C, "HTTP status code reference for CICS Web support," on page 381 has more information about the use of status codes.

The Upgrade header

- Note as a special case that in CICS Web support, protocol upgrading is not supported. This means:
  - For CICS as an HTTP server, it is not possible for an application to take any action in response to an Upgrade header sent by a Web client.
  - For CICS as an HTTP client, the Upgrade header must not be written on requests.

  CICS does not support a switch in HTTP version during a connection, and upgrades in the security layer are not supported.

CICS as an HTTP server: Headers where CICS takes action on receiving an HTTP request

Table 15 shows the action that CICS takes for certain headers on a request received from a Web client.

Table 15. CICS as an HTTP server: CICS actions for headers on an HTTP request

Header received from Web client	Action taken by CICS where response is to be handled by user application program	Action taken by CICS where response is to be provided by static document
Authorization	Passes supplied user ID and password to RACF for verification, and rejects request if these are invalid.	As for application-generated response.
Connection	Carries out Web client's request for connection close after sending response.	As for application-generated response.
Content-Length	CICS requires the Content-Length header on all inbound HTTP/1.1 messages that have a message body. If a message body is present but the header is not provided, or its value is inaccurate, the socket receive for the faulty message or for a subsequent message can produce unpredictable results. For HTTP/1.0 messages that have a message body, the Content-Length header is optional.	Although a message body is not used in processing for a static response, it must still be received from the socket, so the same requirements apply as for an application-generated response.
Content-Type	Parses header to identify media type and character set for code page conversion.	Parses header to identify character set for code page conversion of response.
Expect	Sends 100-Continue response to Web client and waits for remainder of request.	As for application-generated response.
Host	If this header is not present and the client is HTTP/1.1, sends 400 (Bad Request) response to Web client.	As for application-generated response.

Table 15. CICS as an HTTP server: CICS actions for headers on an HTTP request (continued)

Header received from Web client	Action taken by CICS where response is to be handled by user application program	Action taken by CICS where response is to be provided by static document
If-Modified-Since	No action by CICS. User applications could either check for the presence of this header and respond as appropriate, or ignore the header and assume that the application-generated response has been modified.	Document template: Assumes that the response has been modified and sends the requested item. HFS file: Checks modification date and responds according to result of check. Sends 304 response if item has not been modified.
If-Unmodified-Since	If header is present, always sends 412 (Precondition Failed) response to Web client, indicating that the response has been modified since the specified time. (This means that user applications do not have to check for the presence of this header.)	Document template: As for application-generated response, assumes that the response has been modified and sends 412 response. HFS file: Checks modification date and responds according to result of check.
Trailer	Makes individual trailing headers available to application through WEB READ HTTPHEADER command.	Chunked messages are not suitable for a static response.
Transfer-Encoding	For "chunked", receives all chunks and assembles into single message to pass to application. For anything other than "chunked", sends 501 (Not Implemented) response to Web client. The Transfer-Encoding header remains on the message, but it is for information only.	Chunked messages are not suitable for a static response.
Warning	Writes warning text to the TS queue CWBW. If more than 128 characters are used, the warning text is truncated.	As for application-generated response.

CICS as an HTTP client: Headers where CICS takes action on receiving an HTTP response

Table 16 shows the action that CICS takes for certain headers on a response received from a server.

Table 16. CICS as an HTTP client: CICS actions for headers on an HTTP response

Header received from server	Action taken by CICS
Connection	Carries out server's request for connection close after receiving response.
Content-Length	CICS requires the Content-Length header on all inbound HTTP/1.1 messages that have a message body. If a message body is present but the header is not provided, or its value is inaccurate, the socket receive for the faulty message or for a subsequent message can produce unpredictable results. For HTTP/1.0 messages that have a message body, the Content-Length header is optional.
Content-Type	Parses header to identify media type and character set for code page conversion.
Trailer	Makes trailing headers available to application through WEB READ HTTPHEADER command.

Table 16. CICS as an HTTP client: CICS actions for headers on an HTTP response (continued)

Header received from server	Action taken by CICS
Transfer-Encoding	For "chunked", receives all chunks and assembles into single message to pass to application. For anything other than "chunked", sends 501 (Not Implemented) response to Web client. The Transfer-Encoding header remains on the message, but it is for information only.
Warning	Writes warning text to the TS queue CWBW. If more than 128 characters are used, the warning text is truncated.

CICS as an HTTP server: Headers that CICS writes for an HTTP response

Table 17 shows the headers that CICS writes when responding to a request from a Web client, the HTTP versions for which the headers are used, and the source of the information that CICS provides in the header.

Table 17. CICS as an HTTP server: CICS-written headers for an HTTP response

Header written by CICS	HTTP version	Source where response is handled by user application program	Source where response is provided by static document
Connection	1.0 and 1.1	CLOSESTATUS option on WEB SEND command. If no close is specified, and client is HTTP/1.0, Keep-Alive is sent. If close is specified, Connection: close is sent, or for HTTP/1.0 client Keep-Alive is omitted.	Keep-Alive is sent on static responses.
Content-Length (unless chunked transfer-coding is used)	1.0 and 1.1	Where response body is a buffer of data, the length is taken from the FROMLENGTH option on the WEB SEND command. (CICS checks the length that you specify. If it is wrong, CICS sends the response but omits the Connection: Keep-Alive header.) Where response body is a CICS document, the length is calculated by CICS.	Calculated by CICS.
Content-Type	1.0 and 1.1	MEDIATYPE option on WEB SEND command, and character set for response body. (Header is only created when the MEDIATYPE option was specified.)	MEDIATYPE attribute of URIMAP resource definition for request, and character set for response body.
Date	1.0 and 1.1	Current date and time generated by CICS.	Current date and time generated by CICS.
Last-Modified (for static HFS files only)	1.0 and 1.1	**Not provided for dynamic response.** Application should produce this where feasible.	For HFS file: Modification date of file. For document template: Not provided.
Server	1.0 and 1.1	Preset to "IBM_CICS_ Transaction_Server/ 4.1.0(zOS)".	Preset to "IBM_CICS_ Transaction_Server/ 4.1.0(zOS)".
Transfer-Encoding	1.1 only	CHUNKING option on WEB SEND command.	Not used.
WWW-Authenticate	1.0 and 1.1	AUTHENTICATE attribute of TCPIPSERVICE resource definition.	AUTHENTICATE attribute of TCPIPSERVICE resource definition.

CICS as an HTTP client: Headers that CICS writes for an HTTP request

Table 18 shows the headers that CICS writes when an application program sends out a client request to a server, the HTTP versions for which the headers are used, and the source of the information that CICS provides in the header.

Table 18. CICS as an HTTP client: CICS-written headers for an HTTP request

Header written by CICS	**HTTP version**	**Source**
Connection	1.0 and 1.1	CLOSESTATUS option on WEB SEND command. Value of header is selected according to HTTP version of server.
Content-Length (unless chunked transfer-coding is used)	1.0 and 1.1	FROMLENGTH option on WEB SEND command. (CICS checks the length that you specify. If it is wrong, CICS sends the response but omits the Connection: Keep-Alive header.)
Content-Type	1.0 and 1.1	MEDIATYPE option on WEB SEND command, and character set for response body. (Header is only created when the MEDIATYPE option was specified.) Application programs can supply the Content-Type header instead of CICS, if the required header needs to contain spaces or more than 56 characters, and so cannot be specified on the MEDIATYPE option.
Date	1.0 and 1.1	Current date and time generated by CICS, in RFC 1123 format with GMT time.
Expect	1.1 only	ACTION(EXPECT) option on WEB SEND command. This option must only be used if your request has a message body. CICS does not send the header to HTTP/1.0 servers. If CICS does not yet know the server version, specifying the ACTION(EXPECT) option triggers an additional request with the OPTIONS method.
Host	1.0 and 1.1	HOST option on WEB OPEN command.
TE	1.1 only	Always added by CICS when sent to HTTP/1.1 servers, to state that chunked messages and trailers are accepted. (Chunked messages are not sent by HTTP/1.0 servers.) The application program may add further TE headers.
Transfer-Encoding	1.1 only	The first WEB SEND command in a sequence to send a chunked message (CHUNKING option on command indicates chunked transfer-coding). Transfer-Encoding header is written only on first chunk of message.
User-Agent	1.0 and 1.1	Preset to "IBM_CICS_ Transaction_Server/ 4.1.0(zOS)".

Appendix C. HTTP status code reference for CICS Web support

HTTP status codes are provided to clients by a server, to explain the consequence of the client's request. When CICS is an HTTP server, depending on the circumstances, either CICS Web support or the user application program selects an appropriate status code for each response. When CICS is an HTTP client, most status codes received from the server are passed to the user application program for handling.

"Status codes and reason phrases" on page 15 explains how status codes are used in HTTP responses.

For full information about the meaning and correct use of status codes, you should consult the HTTP specification to which you are working. "The HTTP protocol" on page 12 has more information about the HTTP specifications.

This topic provides a brief summary of the HTTP/1.1 status codes as they relate to CICS Web support. When you are selecting status codes to be sent through the Web error programs, or directly from a user application, it is important to check the HTTP specification to which you are working. The HTTP specification provides detailed guidance and requirements about how you should use status codes, such as what should be the content of the response body, and what HTTP headers should be included.

Status codes for responses sent by CICS (when CICS is an HTTP server)

- CICS Web support generates a response to a Web client in the following circumstances:
  - When CICS Web support detects a problem in initial processing of a request from a Web client; for example, if required information is missing from the request, or if the request is sent too slowly and the receive timeout is reached.
  - When an installed URIMAP definition matches the request, but the URIMAP definition or virtual host is disabled, or the resource for a static response cannot be read.
  - When the matching URIMAP definition refers the request to an ATOMSERVICE resource definition, but the ATOMSERVICE definition is disabled, or the CICS resource for the Atom feed cannot be read.
  - When URIMAP matching fails, and the analyzer specified for the TCPIPSERVICE definition is unable to process the request and passes control to a Web error program.
  - When neither the URIMAP definition, nor the analyzer and converter program processing, manages to determine what application program should be executed to service the request.
  - When an abend occurs in the analyzer program, converter program, or user-written application program. This ensures that a response can be returned to the Web client even though processing has failed.
  - When a URIMAP specifies a redirection response.
  - When a Web client is not authorized to access the resources needed to provide the response.

In these situations, CICS selects an appropriate status code and creates a default response. Table 19 on page 383 describes the status codes used by CICS for these purposes. Note that CICS does **not** generate a response in situations where the user-written application program has completed processing successfully and wants to return a response indicating an error; for example, where the client has specified a method not supported for the resource. The user-written application creates the response in this case.

- For most CICS-generated responses with 4xx and 5xx status codes, the response sent to the Web client can be modified by tailoring the user-replaceable Web error programs DFHWBEP and DFHWBERX. CICS-generated responses involving 1xx, 2xx and 3xx status codes cannot be modified. The Web error programs can change the status code, reason phrase, HTTP headers and message body for the response. When you modify the Web error programs, ensure that your selection of status code and response content is made according to the requirements in the HTTP specification to which you are working. Chapter 9, "Web error program," on page 115 explains how to tailor the Web error programs.
- A user application program that responds to a client's request needs to select a suitable status code for the response. The status code can convey the following messages to a Web client:
  - The request has completed as expected.
  - There is an error that prevents fulfilment of the request.
  - The client needs to do something else in order to complete its request successfully. This could involve following a redirection URL, or amending the request so that it is acceptable to the server.

  The status code influences the other content of the response, that is, the message body and HTTP headers. "Sending an HTTP response from CICS as an HTTP server" on page 86 tells you how to assemble and send a response, including a status code and reason phrase.

Status codes for responses received by CICS (when CICS is an HTTP client)

- When CICS is an HTTP client, CICS Web support passes responses with most status codes directly to the user application program for handling. A small number of status codes are handled by CICS and are not returned to the application. If a status code is passed to the application, this indicates that CICS has not taken any action in response to the code, and it is the application's responsibility to check the code and take appropriate action.
- You should design your user application to act appropriately when it receives a message with a status code indicating an error. In particular, you should always 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.

  Check the HTTP specification to which you are working for guidance on what action is appropriate. The HTTP/1.1 specification contains no MUST level requirements that demand further action from the application on receiving a status code, but there are some SHOULD requirements, such as the requirement to follow a redirection.

CICS as an HTTP server: Status codes that CICS provides to Web clients

Table 19 shows the status codes used in situations in which CICS provides a response to a Web client's request. Some of these responses can be tailored by modifying the Web error programs. A user application program may also use many of the status codes listed here.

Some status codes are only appropriate for HTTP/1.1 clients. CICS does not return these status codes to HTTP/1.0 clients.

Table 19. CICS as an HTTP server: Status codes for CICS-generated responses sent to Web clients

Status code and reason phrase provided	Sent to HTTP/1.0 clients?	Situation(s) in which this response is provided	Can be modified in Web error program?
100 Continue	No	Web client sent an Expect header.	No
200 OK	Yes	Delivery of normal response.	No
201 Created	Yes	A new object has been created.	No
301 Moved Permanently	Yes	URIMAP definition specifies a redirection, with attribute REDIRECTTYPE (PERMANENT).	No
302 Found	Yes	URIMAP definition specifies a redirection, with attribute REDIRECTTYPE (TEMPORARY).	No
304 Not Modified	Yes	If-Modified-Since header was used on request, and CICS is able to verify that the static response has not been modified.	Yes
400 Bad Request (some situations: Invalid Request)	Yes	Syntax error in request (such as request line wrongly specified, request incomplete, problem in Atom entry for Atom POST or PUT request). OR Host header is not supplied (HTTP/1.1 only). OR A PUT request without an If-Match header was received. A client that wants to update an object without knowing the current entity tag should specify If-Match: *.	Yes
401 Basic Authentication Error	Yes	User ID and password required for basic authentication. This is determined by security settings for the TCPIPSERVICE definition for the port.	Yes
403 Forbidden (some situations: Client Authentication Error)	Yes	Basic authentication was not successful. OR There is a problem with the client certificate. OR User is not authorized to access a resource that is needed to provide the response, such as an ATOMSERVICE resource definition, alias transaction, CICS command used by a program, or CICS resource containing response data.	Yes
404 Not Found (some situations: Program Not Found, File Not Found)	Yes	The program specified to respond to the request is not found. OR A resource that is needed to provide the response is not found. OR A record is not found within a CICS resource that is used to provide data for an Atom feed.OR An image file is not found.	Yes
408 Request Timeout	No	Receive timeout for request has been exceeded. This is determined by the SOCKETCLOSE attribute in the TCPIPSERVICE definition for the port.	Yes
409 Conflict (some situations: Duplicate resource)	Yes	An existing object already exists with the specified URL, so the new object is not created.	No

Table 19. CICS as an HTTP server: Status codes for CICS-generated responses sent to Web clients (continued)

Status code and reason phrase provided	Sent to HTTP/1.0 clients?	Situation(s) in which this response is provided	Can be modified in Web error program?
412 Precondition Failed	Yes	If-Unmodified-Since header was used on request. OR The entity tag value on the If-Match header does not match the entity tag for the object being updated.	Yes
417 Expectation Failed	No	Expect header received which did not have value "100-continue".	No
500 Internal Server Error (some situations: Resource Error)	Yes	Abend in one of the programs involved with processing the request and providing the response. OR Error reading z/OS UNIX file for a static response.OR Error involving a resource for an Atom feed, such as an error producing XML markup from a resource record for use as Atom entry content.	Yes
501 Method Not Implemented	Yes	Method is not supported by CICS for this HTTP version. (Includes methods that are supported but not in the way the client requests, such as OPTIONS requests that cite a specific resource.) OR Media type for request is "multipart/byteranges", which is not supported. OR Transfer coding for request is other than "chunked". (Note: Connection is closed by CICS.)	Yes
503 Service Unavailable	Yes	A matching URIMAP definition exists, but either it is disabled, or the virtual host of which it is a part is disabled. OR A matching URIMAP definition references an ATOMSERVICE resource definition that is disabled.OR The resource specified in the URIMAP definition or ATOMSERVICE definition for providing response data is disabled.	Yes
505 Version Not Supported	No	HTTP version is higher than 1.1, and method is not recognized for highest version supported by CICS.	Yes

CICS as an HTTP server: Status codes in user applications

Table 20 shows each status code, describes its relevance for a user application, and suggests appropriate actions, in accordance with the recommendations in the HTTP/1.1 specification.

Remember that CICS does not take any specific action that might be implied by these status codes, and that CICS does not generally check their validity against the content of the message. You should ensure that the status codes are correct and that you have taken any necessary action. Ensure that you check the HTTP specification to which you are working, for further information and requirements that apply to each status code.

Table 20. CICS as an HTTP server: Status codes for user-written responses sent to Web clients

Status code and usual reason phrase	Suitable for HTTP/1.0 client?	Situation(s) in which you might provide this response	Effect on message body and HTTP headers (where status code is appropriate for a user application). See HTTP specification for more information.
100 Continue	No	Do not use. CICS handles Expect requests and sends 100-Continue response itself.	

Table 20. CICS as an HTTP server: Status codes for user-written responses sent to Web clients (continued)

Status code and usual reason phrase	Suitable for HTTP/1.0 client?	Situation(s) in which you might provide this response	Effect on message body and HTTP headers (where status code is appropriate for a user application). See HTTP specification for more information.
101 Switching Protocols	No	Do not use. CICS does not support upgrades in HTTP version or security protocol.	
200 OK	Yes	You have fulfilled the request. A normal response.	Provide normal response body.
201 Created	Yes	You have created a new resource. (Use 202 Accepted if the resource has not yet been created.)	Message body content and one or more headers required.
202 Accepted	Yes	You have accepted the request but have not yet processed it, and do not guarantee to process it.	Message body content required.
203 Non-Authoritative Information	No	Do not use. The headers that you supply will give authoritative information.	
204 No Content	Yes	You are not sending a message body, perhaps because you only need to send updated headers.	No message body permitted.
205 Reset Content	No	You want the client to clear the form that initiated the request.	No message body permitted.
206 Partial Content	No	You support byte range requests, and this response fulfils the request.	Normal response body. One or more headers required.
300 Multiple Choices	Yes	You are able to provide more than one version of the resource (for example, documents in different languages).	Message body content and one or more headers required.
301 Moved Permanently	Yes	Not recommended for issuing by user application. Redirection can be managed using the LOCATION and REDIRECTTYPE attributes in the URIMAP definition, so that CICS generates a correct response without calling an application program. REDIRECTTYPE (PERMANENT) selects this status code.	
302 Found	Yes	Not recommended for issuing by user application. When you use a URIMAP definition for redirection, REDIRECTTYPE (TEMPORARY) selects this status code.	
303 See Other	No	You want client to make a GET request for another resource that gives a response (in particular, a response about the outcome of a POST request).	Message body content and one or more headers required.

Table 20. CICS as an HTTP server: Status codes for user-written responses sent to Web clients (continued)

Status code and usual reason phrase	Suitable for HTTP/1.0 client?	Situation(s) in which you might provide this response	Effect on message body and HTTP headers (where status code is appropriate for a user application). See HTTP specification for more information.
304 Not Modified	Yes	The client made a conditional request, and the resource you are providing has not changed. Note that a response that is built dynamically by an application is likely to be modified on every request. For resources that do not change, consider delivering a static response using a URIMAP definition.	No message body permitted. (You can use the DOCTOKEN option to specify a document with no content.) One or more headers required.
305 Use Proxy	No	You want client to go through a named proxy for its request.	One or more headers required.
307 Temporary Redirect	No	Not recommended for issuing by user application. CICS uses the 302 status code, rather than this status code, for URIMAP redirection.	
400 Bad Request	Yes	The client's request contains syntax errors or similar problems, and you cannot process it.	Message body content required.
401 Unauthorized	Yes	Do not use. CICS handles basic authentication process when this is specified in the security settings for the TCPIPSERVICE definition.	
403 Forbidden	Yes	You are refusing the client's request.	Message body content required.
404 Not Found	Yes	You do not have a resource to respond to the request; or you want to refuse the request without explanation; or no other status code is relevant.	Message body content required.
405 Method Not Allowed	No	The client used a method that is not supported for this resource.	Message body content and one or more headers required.
406 Not Acceptable	No	The client made a conditional request using Accept headers, but you do not have a version of the resource that meets their criteria. Note that as an alternative to using this status code, you can send a response which does not meet the conditions.	Message body content required.
407 Proxy Authentication Required	No	Do not use. CICS does not act as a proxy server.	
408 Request Timeout	No	Not recommended for issuing by user application. Timeout should be specified for handling by CICS Web support using the SOCKETCLOSE attribute on the TCPIPSERVICE definition.	
409 Conflict	No	The resource has been changed and the client's request cannot be applied to the resource as it now stands.	Message body content required.
410 Gone	No	The resource is permanently unavailable.	Message body content required.

Table 20. CICS as an HTTP server: Status codes for user-written responses sent to Web clients (continued)

Status code and usual reason phrase	Suitable for HTTP/1.0 client?	Situation(s) in which you might provide this response	Effect on message body and HTTP headers (where status code is appropriate for a user application). See HTTP specification for more information.
411 Length Required	No	Do not use. CICS requires HTTP/1.1 requests to specify the Content-Length header for successful socket receive.	
412 Precondition Failed	No	The client made a conditional request and the conditions were not met.	Message body content required.
413 Request Entity Too Large	No	Not recommended for issuing by user application. Request size limit should be specified for handling by CICS Web support using the MAXDATALEN attribute on the TCPIPSERVICE definition.	
414 Request URI Too Long	No	The client's request URL is too large for your application to process.	Message body content required.
415 Unsupported Media Type	No	The message body sent by the client has a media type or content coding that you do not support.	Message body content required.
416 Requested Range Not Satisfiable	No	The client made a request using the Range header field (but not the If-Range header field), and although you support byte-ranges, that range was not present in the resource.	Message body content and one or more headers required.
417 Expectation Failed	No	Do not use. CICS handles Expect requests.	
500 Internal Server Error	Yes	You cannot handle the request because of an application or system error.	Message body content required.
501 Not Implemented	Yes	The method for the client's request is not supported. This status code should only be issued where the client is HTTP/1.0, or you are using the USER protocol. For the HTTP protocol, during initial processing, CICS rejects any requests with methods that are not recognized. If the method is recognized but does not apply for the resource, 405 Method Not Allowed should be used for HTTP/1.1 clients.	Message body content required.
502 Bad Gateway	Yes	Do not use. CICS does not act as a proxy or gateway.	
503 Service Unavailable	Yes	A user application is unlikely to be in a relevant situation to use this status code, unless it needs to access another application or system which is temporarily unavailable.	Message body content and one or more headers required.
504 Gateway Timeout	No	Do not use. CICS does not act as a proxy or gateway.	
505 HTTP Version Not Supported	No	Do not use. CICS matches HTTP version of response to HTTP version of client's request.	

CICS as an HTTP client: Handling status codes received on responses from servers

Table 21 shows the status codes that you might receive on a response from a server, and suggests appropriate actions, in accordance with the recommendations in the HTTP/1.1 specification. The WEB RECEIVE command returns the status code and status text. Bear in mind that the server might have changed the text of the reason phrase from the text that is suggested in the HTTP specification.

Ensure that you check the HTTP specification to which you are working, for further information and requirements that apply to each status code.

Table 21. CICS as an HTTP client: Handling status codes on responses

Status code and probable reason phrase	**Why would the server send this status code?**	**Suggested action by user application program**
100 Continue	You used the ACTION(EXPECT) option on the WEB SEND command, and the server accepts the full message send.	CICS handles this response by sending message body. User application will not receive this status code.
101 Switching Protocols	Should not be used. Protocol upgrading is not supported by CICS Web support.	User application should not receive this status code.
200 OK	Request is successful. A normal response.	Continue processing the response as planned.
201 Created	You requested creation of a resource and this has been done.	Continue processing the response as planned.
202 Accepted	Server accepts your request but processing has not yet been carried out.	Continue processing the response as planned, but note that any changes you made have not necessarily been committed, and might never be committed.
203 Non-Authoritative Information	Headers relating to message body are not an exact match with those on the server.	Continue processing the response as planned.
204 No Content	There is no message body for the response.	Continue processing the response as planned, but note that there is no body to receive.
205 Reset Content	Server wants you to clear the form that caused the request to be sent.	Clear any form fields that you were using to make the request.
206 Partial Content	You made a request using the Range header field and it was successful.	Continue processing the response as planned.
300 Multiple Choices	Different versions of the resource are available.	Choose your preferred version from the information provided, and make a new request. There might be a Location header containing the URL for the server's preferred choice.
301 Moved Permanently	The resource has moved permanently to a new location.	Make a new request to the URL supplied by the server (probably in the Location header), and use this for all future requests.
302 Found	The resource has moved temporarily to a new location.	Make a new request to the URL supplied by the server (probably in the Location header), but do not use this for future requests.
303 See Other	Server wants you to make a GET request for another resource that gives a response (in particular, a response about the outcome of a POST request).	Make a new request, using the GET method, to the URL supplied by the server (probably in the Location header).

Table 21. CICS as an HTTP client: Handling status codes on responses (continued)

Status code and probable reason phrase	Why would the server send this status code?	Suggested action by user application program
304 Not Modified	You made a conditional request and the resource has not changed.	Refer to your existing stored version of the response for the information, but do not present this to a user as current information, because CICS does not support caching.
305 Use Proxy	Server wants you to use the specified proxy for your request.	Make a new request using the URL supplied by the server (in the Location header).
307 Temporary Redirect	As for 302 Found.	As for 302 Found.
400 Bad Request	Something is wrong with the syntax of your request.	Check the request, make changes and try again.
401 Unauthorized	Server requires authorization; or your supplied authorization has been refused.	See "CICS as an HTTP client: authentication and identification" on page 175.
403 Forbidden	Server refuses your request.	Do not repeat the request. Message body might contain information about why the request was refused.
404 Not Found	Server has not found the requested URL.	Check that the request was specified as you intended. The situation might be temporary, so consider trying again later.
405 Method Not Allowed	You specified a method which is not supported for this resource.	Read the Allow header in the response for a list of supported methods, and make a new request using one of these methods, if wanted.
406 Not Acceptable	You made a request using Accept headers, and the server does not have a version of the resource that meets your criteria.	Examine the message body for information about resources that the server does have, and make a new request for one of these, if wanted.
407 Proxy Authentication Required	A proxy server requires authorization; or your supplied authorization has been refused.	See "CICS as an HTTP client: authentication and identification" on page 175.
408 Request Timeout	Server will not wait any longer for you to complete your request.	Repeat the request, if wanted. Check that your application is not taking a long time to assemble and send the message.
409 Conflict	The resource has been changed and your request cannot be applied to the resource as it now stands.	Examine the message body for information about the cause of the conflict, and make a new request based on this information, if wanted.
410 Gone	The resource is permanently unavailable.	Do not repeat the request in the future.
411 Length Required	Server requires you to supply a Content-Length header.	CICS normally provides that header, unless you are using the USER protocol on the TCPIPSERVICE definition. If that is the case, write the header yourself and make a new request.
412 Precondition Failed	You made a conditional request and the conditions were not met.	Continue processing as planned, noting that any action specified in your request has not been applied.
413 Request Entity Too Large	Your message body is too large for the server to process.	Read the Retry-After header to see if the situation is temporary. You may wait, or reduce the length of the message body, and try again. You might need to open a new connection.
414 Request URI Too Long	Your request URL is too long for the server to process.	Check the request and try again, or abandon the request.

Table 21. CICS as an HTTP client: Handling status codes on responses (continued)

Status code and probable reason phrase	**Why would the server send this status code?**	**Suggested action by user application program**
415 Unsupported Media Type	You sent a message body with a media type or content coding that the server does not support for this resource.	Check the media type that you have specified, and correct and repeat the request if you have made an error.
416 Requested Range Not Satisfiable	You made a request using the Range header field, but that range was not present in the resource.	Read the Content-Range header to see the actual length of the resource, and repeat the request with an appropriate byte range, if wanted.
417 Expectation Failed	You used the ACTION (EXPECT) option on the WEB SEND command, but the server does not accept the full message send.	You may repeat the same request without the ACTION (EXPECT) option, but it will be likely to fail again. Check the request is correctly specified, and correct and repeat the request if you have made an error.
500 Internal Server Error	Server cannot handle the request because of an unexpected error.	The situation might be temporary, so consider trying the request again later.
501 Not Implemented	Server does not support this request method.	Do not repeat the request.
502 Bad Gateway	Your request has gone through a proxy or gateway, which has received an invalid response from another server.	The situation might be temporary, so consider trying the request again later, perhaps avoiding the proxy or gateway if possible.
503 Service Unavailable	Server is temporarily unable to handle the request.	Read the Retry-After header to see if the condition is temporary, and if it is, try again after that time.
504 Gateway Timeout	Your request has gone through a proxy or gateway, which did not receive a timely response from another server.	Repeat the request if wanted, perhaps avoiding the proxy or gateway if possible.
505 HTTP Version Not Supported	Should not be used. CICS Web support sends client requests with HTTP/1.1 as version.	User application should not receive this status code.

Appendix D. HTTP method reference for CICS Web support

HTTP requests include a method, which is a keyword explaining the action that the client wants the server to perform for the material included in the request. CICS Web support implements all the standard request methods defined by the HTTP/1.1 specification, and some additional methods that were accepted in earlier CICS releases.

For detailed guidance on the correct use of methods and the correct actions in response to them, and information on applicable requirements, always consult the HTTP specification to which you are working.

- For requests received from HTTP/1.1 Web clients (when CICS is an HTTP server), the standard methods defined by the HTTP/1.1 specification are accepted. These methods are GET, HEAD, POST, PUT, TRACE, OPTIONS, and DELETE.
- For requests received from HTTP/1.0 Web clients and below (when CICS is an HTTP server), the methods defined by the HTTP/1.0 specification, and some additional methods, are accepted:
  - The methods defined by the HTTP/1.0 specification are GET, HEAD, and POST.
  - The additional methods accepted on HTTP/1.0 requests inbound to CICS are PUT, DELETE, LINK, UNLINK, and REQUEUE.
- For requests made by CICS as an HTTP client:
  - The standard methods defined by the HTTP/1.1 specification can be used. These methods are GET, HEAD, POST, PUT, TRACE, OPTIONS, and DELETE.
  - The LINK, UNLINK, and REQUEUE methods are not supported for this purpose.
  - The version of the request is always given as HTTP/1.1.
  - Some HTTP/1.0 servers may accept methods that are not defined in the HTTP/1.0 specification. An HTTP/1.0 server should return the status code 501 Not Implemented for methods that it cannot accept.
- Message bodies are appropriate for some request methods and inappropriate for others.
  - For CICS as an HTTP server, you should be aware that some clients (particularly user-written clients) might send a message body for a method where it is not appropriate, and you can handle or ignore this as you choose.
  - For CICS as an HTTP client, CICS bars the sending of a message body for methods where it is inappropriate, and requires it for methods where it is appropriate.
- When CICS is an HTTP server, for requests received from a Web client, CICS Web support takes a range of actions in response to the method, depending on the method and the HTTP version of the client.
  - Requests with most methods are passed directly to the application program for handling.
  - CICS automatically returns appropriate responses for the OPTIONS and TRACE methods, without calling a user application program.

  - If a method is not implemented at the HTTP version for the request, CICS returns an error response to the Web client, without calling a user application program.
- In addition to the standard request methods defined in the HTTP specifications, nonstandard request methods, known as extension methods, might be implemented by some servers.
  - For CICS as an HTTP server, CICS Web support does not accept requests with nonstandard methods on the HTTP protocol. (Before CICS Transaction Server for z/OS, Version 4 Release 1, these requests were accepted and processed as non-HTTP.) If you need to receive requests with nonstandard methods, this can be done with the user-defined protocol (USER option on the TCPIPSERVICE definition), where HTTP acceptance checks do not take place.
  - For CICS as an HTTP client, you cannot use nonstandard methods on **EXEC CICS** WEB API commands.

The tables in this reference list the circumstances in which each method may be used. Consult the HTTP specification to which you are working, for more detailed guidance about the methods mentioned in this reference.

CICS as an HTTP server: Handling request methods received from a Web client

Table 22 shows the actions that CICS takes for request methods, and the actions suggested for a user application program. It is important to check the HTTP specification to which you are working, for detailed guidance and any relevant requirements.

Table 22. CICS as an HTTP server: Request methods received from a Web client

Method	CICS action with HTTP/1.0 client	CICS action with HTTP/1.1 client	Message body appropriate on request?	Suitable action by user application program
GET (Request for resource)	Accepted. Request passed to application.	Accepted. Request passed to application.	No	Send resource to the Web client, or send an error response explaining why you cannot do this.
HEAD (Request for response headers)	Accepted. Request passed to application.	Accepted. Request passed to application.	No	Send resource to the Web client exactly as if responding to a GET request for the same resource. CICS removes response body to leave only headers.
POST (Send input data)	Accepted. Request passed to application.	Accepted. Request passed to application.	Yes	Support for method is optional. Extract data (which might be form fields), process it and send a response to the Web client. May also be used for changing or creating a resource, in which case handle as for a PUT request.

Table 22. CICS as an HTTP server: Request methods received from a Web client (continued)

Method	CICS action with HTTP/1.0 client	CICS action with HTTP/1.1 client	Message body appropriate on request?	Suitable action by user application program
PUT (Send new item)	Accepted. Request passed to application.	Accepted. Request passed to application.	Yes	Support for method is optional. If request is valid, create a resource with the specified URL using the content of the message, or replace your existing resource with the content of the message, as appropriate. Send an acknowledgment to the Web client. The HTTP/1.1 specification has detailed requirements for correct operation. **Tip:** This request type is unlikely to be applicable for your CICS Web support implementation. If wanted, it could be fulfilled by creating a URIMAP definition for the specified URL, and storing the resource to be provided as a static response.
TRACE (See request's path and final state)	Rejected with status code 501 Not Implemented. No user application called.	Accepted. CICS responds. No user application called.	No	Not passed to user application. CICS returns response containing request with original headers plus any headers it acquired (such as the Via header).
OPTIONS (Request for information about server)	Rejected with status code 400 Bad request. No user application called.	CICS supports only OPTIONS without a path (OPTIONS with a path are rejected with 405). Note : OPTIONS * accepted. CICS responds. No user application called.	Undefined	Not passed to user application. CICS returns response with basic information (the HTTP version and server software description).
DELETE (Delete resource)	Accepted. Request passed to application.	Accepted. Request passed to application.	No	Support for method is optional. If request is valid, delete your existing resource, and send an acknowledgment to the Web client.
LINK, UNLINK, REQUEUE	Accepted. Request passed to application.	Rejected with status code 501 Not Implemented. No user application called.	Undefined	Use not recommended, as not described in HTTP/1.1 specifications. For compatibility, HTTP/1.0 request is still passed to application.

CICS as an HTTP client: Using methods on requests to a server

Table 23 on page 394 lists the request methods supported by the CICS API for HTTP client requests, and summarizes the correct use of the methods. For guidance on the correct use of each method, and any requirements that apply to an HTTP client using the method, check the HTTP specification to which you are working.

Table 23. CICS as an HTTP client: Request methods sent to a server

Method	Send to HTTP/1.0 server?	Send to HTTP/1.1 server?	Message body on request?	Purpose
GET (Request for resource)	Yes	Yes	No	Obtain a resource from the server.
HEAD (Request for response headers)	Yes	Yes	No	Obtain the headers for a resource from the server. Enables you to check on the nature, status or size of the resource without having to retrieve the whole body.
POST (Send input data)	Yes	Yes	Yes	Send data to a server. For example, form data might be sent in this way. Servers are not required to support this method.
PUT (Send new item)	Might not be supported by server.	Yes	Yes	Create or modify a resource on the server. The URL for your request is the URL that the resource has on the server. The request can be used to update an existing item or to create a new item. Servers are not required to support this method.
TRACE (See request's path and final state)	Might not be supported by server.	Yes	No	Obtain a response showing the final state of your request and the path it took to the server (shown in the Via header). You can see what proxy servers are being used to handle your request. Servers are not required to support this method.
OPTIONS (Request for information about server)	Might not be supported by server.	Yes	Allowed, but no purpose defined for it at present.	Obtain information about the server. Apply the request to the whole server by specifying * (asterisk) as the request path, or specify a full request path to get information about that resource. Servers are not required to support this method.
DELETE (Delete resource)	Might not be supported by server.	Yes	No	Delete a resource on the server. The request URL is the URL of the item to be deleted. Servers are not required to support this method.
LINK, UNLINK, REQUEUE, and extension methods generally	Not permitted. INVREQ response returned and request not sent.	Not permitted. INVREQ response returned and request not sent.	Undefined	Not available on WEB API for CICS as an HTTP client.

Appendix E. Reference information for analyzer programs

This section provides reference information for analyzer programs, including input and output parameters, and responses and reason codes.

Summary of parameters for analyzer programs

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The names of the parameters and constants for analyzer programs, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of CICS.

Language	Parameters file	Constants file
Assembler	DFHWBTDD	DFHWBUCD
C	DFHWBTDH	DFHWBUCH
COBOL	DFHWBTDO	DFHWBUCO
PL/I	DFHWBTDL	DFHWBUCL

These files give language-specific information about the data types of the fields in the COMMAREA. If you use these files, you must specify XOPTS(NOLINKAGE) on the Translator step; if you do not, the compilation fails.

In the following table, the names of the parameters are given in abbreviated form; each name in the table must be prefixed with **wbra_** to give the name of the parameter.

Table 24. Parameters for analyzer programs

Input wbra_	Inout wbra_	Output wbra_
client_ip_address	**alias_tranid**	**application_style**
client_ipv6_address	**converter_program**	**alias_termid**
content_length	**server_program**	**characterset**
eyecatcher	**user_data_length**	**commarea**
function	**userid**	**dfhcnv_key**
hostname_length		**hostcodepage**
hostname_ptr		**reason**
http_version_length		**response**
http_version_ptr		**unescape**
method_ptr		**user_token**
method_length		
querystring_length		
querystring_ptr		
request_header_length		
request_header_ptr		
request_type		
resource_escaped_ptr		
resource_length		
resource_ptr		
server_ip_address		
server_ipv6_address		
urimap		
user_data_ptr		
version		

Parameters for analyzer programs

The names of the parameters for the analyzer program are given with short explanations, including whether the parameters are input only, output only, or input and output parameters.

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

wbra_alias_tranid
(Input and output)

A string of length 4. The transaction ID of the alias transaction that is to cover the remainder of processing for this request. If a URIMAP definition is involved, this string contains the value of the TRANSACTION attribute. If you do not set this field, or if you set it to blanks, CWBA is used.

wbra_alias_termid
(Output only)

A string of length 4. The terminal ID to be used on the START request for the alias transaction that is to cover the remainder of processing for this request.

wbra_characterset
(Output only)

The name of the IANA character set that the client used for the entity body of the request. This information is used for code page conversion of the entity body of the request and the response. If the request is not an HTTP request, this character set is used to translate the entire request and response. **wbra_hostcodepage** must also be supplied.

wbra_client_ip_address
(Input only)

A fullword 32-bit field that specifies the binary IPv4 address of the client, if **wbra_client_ipv6_address** is not specified. **wbra_client_address** does not support IPv6 addresses.

If there is a non-zero value in **wbra_client_address**, this value is used, and any value in **wbra_client_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **wbra_client_address** to allow the value in **wbra_client_ipv6_address** to be used.

wbra_client_ipv6_address
(Input only)

A 16-byte field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **wbra_client_address** is not specified. This field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the client, or the IPv4 address of the client in IPv6 format. For more information on IP address format, see "IP addresses" on page 6.

wbra_commarea
(Ouput only)

The flag to indicate that pre-CICS TS Version 3 compatibility processing is required for a response that uses a non-Web-aware application and a converter program. This flag means that the Web client receives a response identical with the response it received before CICS TS Version 3.

wbra_content_length
(Input only)

A 32–bit binary representation of the entity body length as specified by the Content-Length HTTP header in the received data.

wbra_converter_program
(Input and output)

A string of length 8. The name of the converter program that is used to process the request. If a URIMAP definition is involved, this string contains the value of the CONVERTER attribute. If this field is not set on output, no converter program is called.

wbra_dfhcnv_key
(Output only)

A string of length 8. The name of a conversion template in the DFHCNV table for code page conversion of the entity body for the request and the response. If the request is not an HTTP request, this template is used to translate the entire request and response.

CICS initializes this field to high values. If you use this field to specify a conversion template, the name you choose must be defined in the DFHCNV table, as described in "Upgrading entries in the code page conversion table (DFHCNV)" on page 55. As an alternative, you can set the **wbra_hostcodepage** and **wbra_characterset** fields to specify the pair of code pages to use for code page conversion. If you set **wbra_dfhcnv_key** to nulls or blanks and do not set **wbra_hostcodepage** and **wbra_characterset**, code page conversion is suppressed.

wbra_eyecatcher
(Input only)

A string of length 8. Its value is "`>analyze`".

wbra_function
(Input only)

A code indicating that an analyzer program is being called. The value is 1.

wbra_hostcodepage
(Output only)

The name of a host code page (IBM EBCDIC code page) suitable for the application program that is handling the request. This information is used for code page conversion of the entity body of the request and the response. If the request is not an HTTP request, this code page is used to translate the entire request and response. `wbra_characterset` must also be supplied.

wbra_hostname_length
(Input only)

The length in bytes of the host name specified on the HTTP request. If no host name is specified, the value is undefined.

wbra_hostname_ptr
(Input only)

A pointer to the host name specified on the HTTP request sent by the client. If an absolute URI is used for the request, the host name is taken from the URI. Otherwise the host name is as specified in the Host header for the request. For HTTP/1.1 requests, a host name is required, so this parameter is always passed to the analyzer. For HTTP/1.0 requests, a host name might not be supplied, in which case the value is undefined.

wbra_http_version_length
(Input only)

For an HTTP request, the length in bytes of the string identifying the HTTP version of the client request. If the request is not an HTTP request, the value is zero.

wbra_http_version_ptr
(Input only)

For an HTTP request, a pointer to the string identifying the HTTP version of the client request. If the request is not an HTTP request, the value is undefined.

wbra_method_length
(Input only)

For an HTTP request, the length in bytes of the string identifying the method specified in the HTTP request. If the request is not an HTTP request, the value is zero.

wbra_method_ptr
(Input only)

For an HTTP request, a pointer to the method specified in the HTTP request. If the request is not an HTTP request, the value is undefined.

wbra_querystring_length
(Input only)

The length in bytes of the query string specified on the HTTP request. If no query string was sent, the value is undefined.

wbra_querystring_ptr
(Input only)

A pointer to the query string specified on the HTTP request sent by the client. If no query string was sent, the value is undefined.

wbra_reason
(Output only)

The reason code returned by the analyzer program. See "Responses and reason codes" on page 401.

wbra_request_header_length
(Input only)

For an HTTP request, the length of the first HTTP header in the HTTP request. If the request is not an HTTP request, the value is zero.

wbra_request_header_ptr
(Input only)

For an HTTP request, a pointer to the first HTTP header in the HTTP request. The other HTTP headers follow this one in the request buffer. If the request is not an HTTP request, the value is undefined.

wbra_request_type
(Input only)

If this request is an HTTP request, the value is WBRA_REQUEST_HTTP. If it is not an HTTP request, the value is WBRA_REQUEST_NON_HTTP.

wbra_resource_escaped_ptr
(Input only)

For an HTTP request, a pointer to a copy of the HTTP headers for the request that have not been unescaped; that is, are still in their escaped form.

wbra_resource_length
(Input only)

For an HTTP request, the length in bytes of the path component of the URL. If the request is not an HTTP request, the value is zero.

wbra_resource_ptr
(Input only)

For an HTTP request, a pointer to the path component of the URL. If a URIMAP definition is involved, this pointer contains the value of the PATH attribute. If the request is not an HTTP request, the value is undefined.

wbra_response
(Output only)

The response value produced by the analyzer program. See "Responses and reason codes" on page 401.

wbra_server_ip_address
(Input only)

A fullword 32-bit field that specifies the binary IPv4 address of the HTTP server, if **wbra_server_ipv6_address** is not specified. **wbra_server_address** does not support IPv6 addresses.

If there is a non-zero value in **wbra_server_address**, this value is used, and any value in **wbra_server_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **wbra_server_address** to allow the value in **wbra_server_ipv6_address** to be used.

wbra_server_ipv6_address
(Input only)

A 16-byte field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **wbra_server_address** is not specified. This field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the server, or the IPv4 address of the server in IPv6 format. For more information on IP address format, see "IP addresses" on page 6.

wbra_server_program
(Input and output)

A string of length 8. The name of a CICS application program that is to process the request. If a URIMAP definition is involved, this string contains the value of the PROGRAM attribute. The program name is passed to any converter program specified in **wbra_converter_program**. If you do not set this field, the value passed is nulls. The program name must be set here or by the converter program; otherwise, no CICS application program is called.

wbra_unescape
(Output only)

- To specify that data is to be passed to the CICS application program in its unescaped form, set this parameter to WBRA_UNESCAPE_REQUIRED.
- To specify that data is be passed to the application in its escaped form, set this parameter to WBRA_UNESCAPE_NOT_REQUIRED. This value is the default.

  Also set the parameter to WBRA_UNESCAPE_NOT_REQUIRED if your analyzer has converted the data to its escaped form.

wbra_urimap
(Input only)

The name of any matching URIMAP definition that is involved in the processing path for the request. If this field is nonblank, the CICS-supplied default analyzer DFHWBADX returns without processing the path component of the URL.

wbra_user_data_length
(Input and output)

A 15-bit integer, representing the length of the entity body in the HTTP request. If the request is non-HTTP, this value is the length of the request. The length passed to the analyzer includes any trailing carriage return and line feed (CRLF) characters that might delimit the end of the entity body. If the analyzer reduces the length of the entity body, the newly redundant bytes are replaced by null characters, X'00'. The modified value is passed to the CICS business logic interface in field **wbbl_user_data_length**, and to the converter program in field **decode_user_data_length**.

wbra_user_data_ptr
(Input only)

For an HTTP request, a pointer to the entity body in the HTTP request. If the request is not an HTTP request, this pointer is to the request.

wbra_user_token
(Output only)

A 64-bit token that is passed to the converter program as **decode_user_token**. If you do not set this field, the value passed is null. If there is no converter program for this request, the value is ignored.

wbra_userid
(Input and output)

A string of length 8. On input, this string contains a user ID supplied by the client (using basic authentication or client certificate authentication), or, if a URIMAP definition is involved, the value of the USERID attribute, if specified. On output, it contains the user ID that is used for the alias transaction, which can be the supplied user ID or a user ID chosen by the analyzer program. If this field is blank or null on output, the CICS default user ID is used.

wbra_version
(Input only)

A halfword binary number that indicates which version of the parameter list is currently being used. It is set using the constant value **wbra_current_version**.

Responses and reason codes

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

An analyzer program must return one of the values shown in the table in **wbra_response**.

Symbolic value	Numeric value	Explanation
URP_OK	0	The alias transaction is started.
URP_EXCEPTION	4	The alias transaction is not started. Web attach processing writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, an error response is sent to the Web client. The default status code is 400 (Bad request), and this can be configured using the user-replaceable Web error program DFHWBEP. If the request is not an HTTP request, no response is sent, and the socket is closed.
URP_INVALID	8	The alias transaction is not started. The server controller writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, an error response is sent to the Web client. The default status code is 400 (Bad request), and this can be configured using the user-replaceable Web error program DFHWBEP. If the request is not an HTTP request, no response is sent, and the socket is closed.

Symbolic value	Numeric value	Explanation
URP_DISASTER	12	The alias transaction is not started. CICS writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, an error response is sent to the Web client. The default status code is 400 (Bad request), and this can be configured using the user-replaceable Web error program DFHWBEP. If the request is not an HTTP request, no response is sent, and the socket is closed.

If you return any other value in **wbra_response**, the server controller writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, a message with status code 400 (Bad request) is sent to the Web client. If the request is not an HTTP request, no response is sent, and the socket is closed.

You may supply a 32-bit reason code in **wbra_reason** to provide further information in error cases. CICS Web support does not take any action on the reason code returned by an analyzer program, but the user-replaceable Web error program DFHWBEP can use it to decide how to modify the default response. The reason code is output in any trace entry that results from the invocation of an analyzer program, and in message DFHWB0523.

Appendix F. Reference information for converter programs

This section provides: reference information for the **Decode** function of a converter program, and, reference information for the **Encode** function of a converter program.

The names of the parameters and constants in the COMMAREA passed to a converter program, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of CICS Web support. The files for the various languages are listed in the following table.

Language	Parameters file	Constants file
Assembler	DFHWBCDD	DFHWBUCD
C	DFHWBCDH	DFHWBUCH
COBOL	DFHWBCDO	DFHWBUCO
PL/I	DFHWBCDL	DFHWBUCL

These files give language-specific information about the data types of the fields in the COMMAREA. If you use these files you must specify XOPTS(NOLINKAGE) on the Translator step; failure to do this causes the compile to fail.

Parameter list for converter program decode function

If the analyzer program, URIMAP definition, or caller of the CICS business logic interface specified a converter program name for the request, **decode** is called before the user-written application program that is to provide data for the request.

Summary of parameters

In the following table, the names of the parameters are given in abbreviated form; each name in the table must be prefixed with **decode_** to give the name of the parameter.

Table 25. Parameters for decode

Input decode_	Inout decode_	Output decode_
client_address **client_ipv6_address** **client_address_string** **client_ipv6_address_string** **eyecatcher** **entry_count** **function** **http_version_length** **http_version_ptr** **method_length** **method_ptr** **request_header_length** **request_header_ptr** **resource_length** **resource_ptr** **user_data_length** **user_data_ptr** **version** **volatile**	**data_ptr** **input_data_len** **server_program** **user_token**	**output_data_len** **reason** **response**

Parameters

decode_client_address
(Input only)

A fullword 32-bit field that must be set to the binary IPv4 address of the client, if **decode_client_ipv6_address** is not specified. **decode_client_address** does not support IPv6 addresses.

If there is a non-zero value in **decode_client_address**, this value is used, and any value in **decode_client_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **decode_client_address** to allow the value in **decode_client_ipv6_address** to be used.

decode_client_ipv6_address
(Input only)

A 16-byte field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **decode_client_address** is not specified. This field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the client, or the IPv4 address of the client in IPv6 format. For more information on IP address format, see "IP addresses" on page 6.

decode_client_address_string
(Input only)

The IPv4 address of the client in dotted decimal format.

decode_client_ipv6_address_string
(Input only)

The IP address of the client in dotted decimal format for IPv4 addresses or in colon hexadecimal format for IPv6 addresses. This field can be up to 39 bytes in length.

decode_data_ptr
(Input and output)

On input, a pointer to the request from the client, which might have been modified by the analyzer program, or, if this call is a loop back from the **encode** converter function, a pointer to the response data of **encode_data_ptr**.

On output, pointer to the COMMAREA to be passed to the user-written application program. Do not modify this parameter when **decode_volatile** has a value of **0**.

decode_entry_count
(Input only)

A count to say how many times the **decode** converter has been entered for the current Web request.

decode_eyecatcher
(Input only)

A string of length 8. Its value for **decode** is "`>decode `".

decode_function
(Input only)

A halfword code set to the constant value **URP_DECODE**, indicating that **decode** is being called.

decode_http_version_length
(Input only)

The length in bytes of the string identifying the HTTP version supported by the client. If the request is not an HTTP request, or **decode_entry_count** is greater than 1, the value is zero.

decode_http_version_ptr
(Input only)

A pointer to the string identifying the HTTP version supported by the client. If the analyzer modified this part of the request, the changes are visible here. If **decode_http_version_length** is zero, the value is undefined.

decode_input_data_len
(Input and output)

On input, the length in bytes of the request data pointed to by decode_data_ptr.

decode_method_length
(Input only)

The length in bytes of the method specified in the HTTP request. If the request is not an HTTP request, or **decode_entry_count** is greater than 1, the value is zero.

decode_method_ptr
(Input only)

A pointer to the method specified in the HTTP request. If the analyzer modified this part of the request, the changes are visible here. If **decode_method_length** is zero, the value is undefined.

decode_output_data_len
(Output only)

The length in bytes of the COMMAREA that is to be passed to the user-written application program, as indicated in the pointer **decode_data_ptr**. The default value if this output is not set is 32 KB.

decode_reason
(Output only)

A reason code; see "Responses and reason codes" on page 407.

decode_request_header_length
(Input only)

The length of the first HTTP header in the HTTP request. If the request is not an HTTP request, or **decode_entry_count** is greater than 1, the value is zero.

decode_request_header_ptr
(Input only)

A pointer to the first HTTP header in the HTTP request. If the analyzer program modified this part of the request, the changes are visible here. If **decode_request_header_length** is zero, the value is undefined.

decode_resource_length
(Input only)

The length in bytes of the path component of the URL in the HTTP request. If the request is not an HTTP request, or **decode_entry_count** is greater than 1, the value is zero.

decode_resource_ptr
(Input only)

A pointer to the path component of the URL in the HTTP request. If the analyzer program modified this part of the request, the changes are visible here. If **decode_resource_length** is zero, the value is undefined.

decode_response
(Output only)

A response; see "Responses and reason codes" on page 407.

decode_server_program
(Input and output)

A string of length 8. On input, the value supplied by the analyzer in **wbra_server_program**, or the value supplied by the caller of the CICS business logic interface. On output, the name of the user-written application program that is to service the request. The application program name must be set here or in the analyzer program; if it is not set, no application program is called.

decode_user_data_length
(Input only)

The length in bytes of the entity body for this HTTP request. If the analyzer program modified this value, the modified value is visible here. If there is no entity body in the request, the length is zero. If the request is not an HTTP request, the value is the length of the request. If **decode_entry_count** is greater than 1, the value is zero.

decode_user_data_ptr
(Input only)

A pointer to any entity body for this HTTP request. If the analyzer modified this part of the request, the changes are visible here. If the request has no entity body, the pointer is zero. If the request is not an HTTP request, this pointer has the same value as **decode_data_ptr**. If **decode_entry_count** is greater than 1, the value is undefined.

decode_user_token
(Input and output)

A 64-bit token. On input, the user token supplied by the analyzer as **wbra_user_token**, or the user token supplied by the caller of the CICS business logic interface. On output, a token that is passed to **Encode** as **encode_user_token**.

decode_version
(Input)

A halfword binary number that indicates which version of the parameter list is currently being used. It is set using the constant value **decode_current_version**.

decode_volatile
(Input)

A single-character code indicating whether the data area pointed to by **decode_data_ptr** can be replaced. Possible values are as follows:

0 The area is part of another COMMAREA and cannot be replaced.

1 The storage pointed to by **decode_data_ptr** can be freed and replaced by a work area of a different size.

Responses and reason codes

You must return one of the following values in **decode_response**:

Symbolic value	Numeric value	Explanation
URP_OK	0	Processing continues. If a CICS application program is requested, it is run. If not, processing continues with the **encode** function of the converter program.

Symbolic value	Numeric value	Explanation
URP_EXCEPTION	4	The action taken depends on the reason code: • **CICS Web support** – 1 (URP_SECURITY_FAILURE). CICS writes an exception trace entry (trace point 455A), and issues a message (DFHWB0121). If the request is an HTTP request, status code 403 is sent to the Web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. – 2 (URP_CORRUPT_CLIENT_DATA). CICS writes an exception trace entry (trace point 4559), and issues a message (DFHWB0121). If the request is an HTTP request, status code 400 is sent to the Web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. – Any other value. CICS writes an exception trace entry (trace point 455B), and issues a message (DFHWB0121). If the request is an HTTP request, status code 501 is sent to the Web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. • **CICS business logic interface** – 2 (URP_CORRUPT_CLIENT_DATA). The CICS business logic interface writes an exception trace entry (trace point 4556), issues a message (DFHWB0120), and returns a response of 400 to its caller. – Any other value. CICS writes an exception trace entry (trace point 455B), issues a message (DFHWB0121), and returns a response of 501 to its caller. The CICS application program and the **encode** function of the converter program do not run.
URP_INVALID	8	The CICS application program and the **encode** function of the converter program do not run. • **CICS Web support** – CICS writes an exception trace entry (trace point 455C), and issues a message (DFHWB0121). If the request is an HTTP request, status code 501 is sent to the Web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. • **CICS business logic interface** – CICS writes an exception trace entry (trace point 455C), issues a message (DFHWB0121), and returns a response of 501 to its caller.

Symbolic value	Numeric value	Explanation
URP_DISASTER	12	The CICS application program and the **encode** function of the decoder do not run. • **CICS Web support** – CICS writes an exception trace entry (trace point 455D), and issues a message (DFHWB0121). If the request is an HTTP request, status code 501 is sent to the Web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. • **CICS business logic interface** – CICS writes an exception trace entry (trace point 455D), issues a message (DFHWB0121), and returns a response of 501 to its caller.
any other value		The CICS application program and the **encode** function of the decoder do not run. • **CICS Web support** – CICS writes an exception trace entry (trace point 455E), and issues a message (DFHWB0121). If the request is an HTTP request, status code 500 is sent to the Web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. • • **CICS business logic interface** – CICS writes an exception trace entry (trace point 455E), issues a message (DFHWB0121), and returns a response of 501 to its caller.

You can supply a 32-bit reason code in **decode_reason** to provide further information in error cases. Neither CICS Web support nor the CICS business logic interface takes any action on the reason code returned by **decode**, except as indicated above under URP_EXCEPTION. The reason code is included in any trace entry that results from the invocation of **decode**.

Parameter list for converter program encode function

Summary of parameters

In the following table, the names of the parameters are given in abbreviated form: each name in the table must be prefixed with **encode_** to give the name of the parameter.

Table 26. Parameters for Encode

Input encode_	Inout encode_	Output encode_
eyecatcher **entry_count** **function** **input_data_len** **user_token**	**data_ptr**	**reason** **response**

Function

If the analyzer program, or the caller of the CICS business logic interface, specified a converter program name for the request, **Encode** is called after the user-written application program has ended. It constructs the response using the data in the COMMAREA returned by the application program.

Parameters

encode_data_ptr
(Input and output)

On input, this is a pointer to the COMMAREA returned by the CICS application program. If no application program was called, this is a pointer to the COMMAREA created by the **Decode** function of the converter program.

On output, if the converter program has constructed the HTTP response manually in a buffer of storage for CICS to send to the Web client, this is a pointer to the buffer containing the response. You must ensure that the pointer points to a valid location, or results can be unpredictable. The buffer must be doubleword aligned. The first four bytes must be a 32-bit unsigned number specifying the length of the buffer. (In COBOL, specify this as PIC 9(8) COMP.) The rest of the buffer is the response.

If the converter program has used **EXEC CICS** WEB API commands to send the response instead, CICS ignores and discards any block of storage indicated by this pointer. In this situation, the pointer can be left as a pointer to the COMMAREA returned by the CICS application program; its setting does not matter.

Do not use this field as output when the converter was called from a CICS business logic interface that was called in offset mode.

encode_entry_count
(Input only)

A count to say how many times the **Encode** function of the converter program has been entered for the current Web request.

encode_eyecatcher
(Input only)

A string of length 8. Its value for **Encode** is "`>encode `".

encode_function
(Input only)

A halfword code set to the constant value **URP_ENCODE**, indicating that **Encode** is being called.

encode_input_data_len
(Input only)

The length of the COMMAREA as specified by **Decode** in **decode_output_data_len**.

encode_reason
(Output only)

A reason code (see "Responses and reason codes" on page 411).

encode_response
(Output only)

A response (see "Responses and reason codes").

encode_user_token
(Input only)

The 64-bit token output by the **Decode** function as **decode_user_token**.

encode_version
(Input)

A single-character parameter list version identifier, which changes whenever the layout of the parameter list changes. Its value can be either binary zero (X'00'), indicating a pre-CICS TS 1.3 version parameter list, or a character zero (X'F0'), indicating a CICS TS 1.3 or later version parameter list.

encode_volatile
(Input)

A single-character code indicating whether the data area pointed to by **encode_data_ptr** can be replaced. Possible values are:

0 The area is part of another COMMAREA and cannot be replaced.

1 The storage pointed to by **encode_data_ptr** can be freed and replaced by a different size work area.

Responses and reason codes

You must return one of the following values in **encode_response**:

Symbolic value	Numeric value	Explanation
URP_OK	0	The response in the buffer pointed to by **encode_data_ptr** is sent to the client, unless the **EXEC CICS** WEB API commands have already been used to send a response.
URP_DISASTER	12	**CICS Web support** • CICS writes an exception trace entry (trace point 455D), and issues a message (DFHWB0122). If the request is an HTTP request, status code 501 is sent to the web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. **CICS business logic interface** • CICS writes an exception trace entry (trace point 455D), issues a message (DFHWB0122), and returns a response of 501 to its caller.
URP_OK_LOOP	16	CICS loops back to the start of the **Decode** function. The value stored in **encode_user_token** is copied to **decode_user_token** for the **Decode** converter function to use.

Symbolic value	Numeric value	Explanation
any other value		**CICS Web support** • CICS writes an exception trace entry (trace point 455E), and issues a message (DFHWB0122). If the request is an HTTP request, status code 501 is sent to the web client. If the request is not an HTTP request, no response is sent, and the TCP/IP socket is closed. **CICS business logic interface** • CICS writes an exception trace entry (trace point 455E), issues a message (DFHWB0122), and returns a response of 501 to its caller.

You can supply a 32-bit reason code in **encode_reason** to provide further information in error cases. Neither CICS Web support nor the CICS business logic interface takes any action on the reason code returned by the **Encode** function. The reason code is output in any trace entry that results from the invocation of the **Encode** function.

Appendix G. Reference information for DFHWBBLI, CICS business logic interface

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The business logic interface allows callers to specify what presentation logic is to be executed before and after a CICS application program. It has two modes of operation:

- Pointer mode: the input data for **Decode** is in storage allocated separately from the COMMAREA for the business logic interface. The COMMAREA contains a pointer (**wbbl_data_ptr**) to the input data for **Decode**. When the call to the business logic interface ends, the output from **Encode** is in storage allocated separately from the COMMAREA for the business logic interface, and the COMMAREA contains a pointer (**wbbl_outdata_ptr**) to the output from **Encode**.
- Offset mode: the input data for **Decode** is part of the COMMAREA for the business logic interface. The COMMAREA contains the offset (**wbbl_data_offset**) of the input data for **Decode**. When the call to the business logic interface ends, the output from **Encode** is part of the COMMAREA for the business logic interface, and the COMMAREA contains the offset (**wbbl_outdata_offset**) of the output from **Encode**.

The caller of the business logic interface uses **wbbl_mode** to indicate which mode of operation is to be used.

For information about writing a converter for the business logic interface, see "Writing a converter program" on page 140.

Summary of parameters

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The names of the parameters and constants, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of CICS Web support. These files give language-specific information about the data types of the fields in the COMMAREA.

The files for the various languages are as follows:

Language	File
Assembler	DFHWBBLD
C	DFHWBBLH
COBOL	DFHWBBLO
PL/I	DFHWBBLL

Table 27. Parameters for the business logic interface

Input	Output
wbbl_client_address **wbbl_client_ipv6_address** **wbbl_client_address_length** **wbbl_client_ipv6_address_length** **wbbl_client_address_string** **wbbl_client_ipv6_address_string** **wbbl_converter_program_name** **wbbl_eyecatcher** **wbbl_header_length** **wbbl_header_offset** **wbbl_http_version_length** **wbbl_http_version_offset** **wbbl_indata_length** **wbbl_indata_offset** **wbbl_indata_ptr** **wbbl_length** **wbbl_method_length** **wbbl_method_offset** **wbbl_mode** **wbbl_prolog_size** **wbbl_resource_length** **wbbl_resource_offset** **wbbl_server_address** **wbbl_server_ipv6_address** **wbbl_server_program_name** **wbbl_ssl_keysize** **wbbl_status_size** **wbbl_user_token** **wbbl_user_data_length** **wbbl_vector_size** **wbbl_version**	**wbbl_outdata_length** **wbbl_outdata_offset** **wbbl_outdata_ptr**

Programs that were written to use an earlier version of the CICS business logic interface (DFHWBA1) are supported through a compatibility interface, which calls DHWBBLI.

Parameters for the business logic interface, DFHWBBLI

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

Names and descriptions of the input and output parameters for the business logic interface are listed, prefixed with **wbbl_**.

Before inserting the inputs into the COMMAREA, you must clear it to binary zeros.

wbbl_eyecatcher
(Input only)

A 14-character field that must be set to the string **>DFHWBBLIPARMS**.

wbbl_client_address
(Input only)

A fullword 32-bit field that must be set to the binary IPv4 address of the client, if **wbbl_client_ipv6_address** is not specified. **wbbl_client_address** does not support IPv6 addresses.

If there is a non-zero value in **wbbl_client_address**, this value is used, and any value in **wbbl_client_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **wbbl_client_address** to allow the value in **wbbl_client_ipv6_address** to be used.

wbbl_client_ipv6_address
(Input only)

A 16-byte field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **wbbl_client_address** is not specified. This field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the client, or the IPv4 address of the client in IPv6 format. For more information on IP address format, see "IP addresses" on page 6.

wbbl_client_address_length
(Input only)

A 1-byte binary field that must be set to the length of **wbbl_client_address_string**.

wbbl_client_ipv6_address_length
(Input only)

A 1-byte binary field that must be set to the length of **wbbl_client_ipv6_address_string**.

wbbl_client_address_string
(Input only)

A string of up to 15 characters that are the dotted decimal representation of **wbbl_client_address**, padded on the right with binary zeros. Use **wbbl_client_ipv6_address_string** instead of **wbbl_client_address_string** for all new programs.

wbbl_client_ipv6_address_string
(Input only)

A string of up to 39 characters that are the colon hexadecimal or dotted decimal representation of **wbbl_client_ipv6_address**, padded on the right with binary zeros.

wbbl_converter_program_name
(Input only)

The 8-character name of the program to be used for converter DECODE and ENCODE functions.

wbbl_header_length
(Input only)

A fullword binary number that must contain the length of the HTTP headers associated with this request.

wbbl_header_offset
(Input only)

A fullword binary number that must contain the offset, from the start of the request data, of the HTTP headers associated with this request.

wbbl_http_version_length
(Input only)

A fullword binary number that must contain the length of the version of the HTTP protocol to be used to process the request.

wbbl_http_version_offset
(Input only)

A fullword binary number that must contain the offset of the version of the HTTP protocol to be used to process the request.

wbbl_indata_length
(Input only)

A fullword binary number that must be set to the length of the data located by **wbbl_indata_ptr** or **wbbl_indata_offset**. If the analyzer modified the data length value, it is visible here. If the request is not an HTTP request, do not set this field.

wbbl_indata_offset
(Input only)

If **wbbl_mode** is "O" or "D", this is the offset, from the start of the parameter list, of the HTTP request data to be passed to the application.

wbbl_indata_ptr
(Input only)

If **wbbl_mode** is "P", this field is the address of the HTTP request data to be passed to the application.

wbbl_length
(Input only)

A halfword binary number that must be set to the total length of the BLI parameter list.

wbbl_method_length
(Input only)

A fullword binary number that must contain the length of the HTTP method to be used to process the request. The method is one of: GET, POST, HEAD, PUT, DELETE, LINK, UNLINK, or REQUEUE.

wbbl_method_offset
(Input only)

A fullword binary number that must contain the offset, from the start of the request data, of the HTTP method to be used to process the request. The method is one of: GET, POST, HEAD, PUT, DELETE, LINK, UNLINK, or REQUEUE.

wbbl_mode
(Input only)

A single character that indicates the addressing mode for **wbbl_indata** and **wbbl_outdata**. It must be set to "P" to indicate that these values are pointers or to "O" to indicate that these values are offsets from the start of the parameter list.

wbbl_outdata_length
(Input only)

The fullword binary field in which DFHWBBLI returns the length of the response data located by **wbbl_outdata_ptr** or **wbbl_outdata_offset**.

wbbl_outdata_offset
(Input only)

If **wbbl_mode** is "O" or "D", this is the fullword in which DFHWBBLI returns the offset, from the start of the parameter list, of the response data from the application. This address in not necessarily the same as **wbbl_indata_offset**.

wbbl_outdata_ptr
(Input only)

If **wbbl_mode** is "P", this field is the fullword address in which DFHWBBLI returns the address of the response data from the application. This address in not necessarily the same as **wbbl_indata_ptr**.

wbbl_prolog_size
(Input only)

A halfword binary number that must be set to 56; that is, the length of the **wbbl_prolog** substructure.

wbbl_resource_length
(Input only)

A fullword binary number that must contain the length of the URI resource that is being requested; that is, the non-network part of the URL, starting at the first / character in the URL.

wbbl_resource_offset
(Input only)

A fullword binary number that must contain the offset, from the start of the request data, of the URI resource that is being requested; that is, the non-network part of the URL, starting at the first / character in the URL.

wbbl_response
(Input only)

A fullword binary field in which DFHWBBLI returns its response code.

wbbl_server_address
(Input only)

A fullword 32-bit field that must be set to the binary IPv4 address of the server, if **wbbl_server_ipv6_address** is not specified. **wbbl_server_address** does not support IPv6 addresses.

If there is a non-zero value in **wbbl_server_address**, this value is used, and any value in **wbbl_server_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **wbbl_server_address** to allow the value in **wbbl_server_ipv6_address** to be used.

wbbl_server_ipv6_address
(Input only)

A 16-byte field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **wbbl_server_address** is not specified. This field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the server, or the IPv4 address of the server in IPv6 format. For more information on IP address format, see “IP addresses” on page 6.

wbbl_server_program_name
(Input only)

The 8-character name of the CICS application program that is to be used to process the request and produce the response.

wbbl_ssl_keysize
(Input only)

The size of the encryption key negotiated during the SSL handshake, if secure sockets layer is being used. It contains zero if SSL is not being used.

wbbl_status_size
(Input only)

A 1-byte binary field that must be set to the length of the **wbbl_status** substructure.

wbbl_user_data_length
(Input only)

A fullword binary number that must be set to the length of the entity body. If the analyzer modified the length value, it is visible here. If the request is not an HTTP request, do not set this field.

wbbl_user_token
(Input only)

An 8-character field in which the caller of DFHWBBLI can pass data which identifies the current conversational state with the client. It is usually set to the first eight characters of the **query-string** portion of the URL; that is, any data following a question mark (?).

wbbl_vector_size
(Input only)

A halfword binary number that must be set to 64 (that is, the length of the **wbbl_vector** substructure.

wbbl_version
(Input only)

A halfword binary number that indicates which version of the BLI parameter list is currently being used. It is set using the constant value **wbbl_current_version**.

Business logic interface responses

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

One of the following values is returned in **wbbl_response**. These values correspond to the intended HTTP responses to be sent to an HTTP client.

400 One of the converter functions returned a URP_EXCEPTION response with a reason URP_CORRUPT_CLIENT_DATA. The business logic interface writes an exception trace entry (trace point 4556) and issues a message (DFHWB0120).

403 A LINK command to the program specified in **wbbl_server_program_name** received a NOTAUTH response. The business logic interface writes an exception trace entry (trace point 4556) and issues a message (DFHWB0120).

404 A LINK command to the program specified in

wbbl_server_program_name received a PGMIDERR response. The business logic interface writes an exception trace entry (trace point 4556) and issues a message (DFHWB0120).

500 One of the following occurred:

- The business logic interface detected an abend. A message that depends on the program that abended is issued. For the program specified in **wbbl_server_program_name**, the message is DFHWB0125. For the **Encode** function of the converter, the message is DFHWB0126. For the **Decode** function of the converter, the message is DFHWB0127. For any other program, the message is DFHWB0128. In any case an exception trace entry (trace point 4557) is written.
- A LINK command to the program specified in **wbbl_server_program_name** received an INVREQ or a LENGERR or an unexpected response. The business logic interface writes an exception trace entry (trace point 4556) and issues a message (DFHWB0120).

501 One of the following occurred:

- **Decode** returned a response of URP_EXCEPTION with an undefined reason code. The business logic interface writes an exception trace entry (trace point 455B) and issues a message (DFHWB0121).
- **Decode** returned a response of URP_INVALID. The business logic interface writes an exception trace entry (trace point 455C) and issues a message (DFHWB0121).
- **Decode** returned a response of URP_DISASTER. The business logic interface writes an exception trace entry (trace point 455D) and issues a message (DFHWB0121).
- **Decode** returned an undefined response. The business logic interface writes an exception trace entry (trace point 455E) and issues a message (DFHWB0121).
- **Encode** returned a response of URP_EXCEPTION with an undefined reason code. The business logic interface writes an exception trace entry (trace point 455B) and issues a message (DFHWB0122).
- **Encode** returned a response of URP_INVALID. The business logic interface writes an exception trace entry (trace point 455C) and issues a message (DFHWB0122).
- **Encode** returned a response of URP_DISASTER. The business logic interface writes an exception trace entry (trace point 455D) and issues a message (DFHWB0122).
- **Encode** returned an undefined response. The business logic interface writes an exception trace entry (trace point 455E) and issues a message (DFHWB0122).

503 One of the following occurred:

- A LINK command to the program specified in **wbbl_server_program_name** received a TERMERR response. The business logic interface writes an exception trace entry (trace point 4555) and issues a message (DFHWB0120).
- A LINK command to the program specified in **wbbl_server_program_name** received a SYSIDERR or ROLLEDBACK response. The business logic interface writes an exception trace entry (trace point 4556) and issues a message (DFHWB0120).

Appendix H. Reference information for DFHWBEP, Web error program

Attention: This topic contains Product-sensitive Programming Interface and Associated Guidance Information.

The names of the parameters and constants in the parameter list passed to the Web error program DFHWBEP, translated into appropriate forms for the programming languages supported, are listed in the following table.

Language	Parameters file
Assembler	DFHWBEPD
C	DFHWBEPH
COBOL	DFHWBEPO
PL/I	DFHWBEPL

Parameters

All DFHWBEP parameters are input only, except **wbep_response_ptr** and **wbep_response_len**, which are input and output, and **wbep_suppress_abend** and **wbep_close_conn**, which are output only.

wbep_abend_code
(Input only)

The 8-character abend code associated with this exception.

wbep_activity
(Input only)

The type of processing that was in progress when the error occurred. 0 indicates server processing and 2 indicates pipeline processing.

wbep_analyzer_reason
(Input only)

The reason code returned by the analyzer program, if invoked.

wbep_analyzer_response
(Input only)

The response code returned by the analyzer program, if invoked.

wbep_client_address
(Input only)

A 15–character field that must be set to the binary IPv4 address of the client, if **wbep_client_ipv6_address** is not specified. **wbep_client_address** does not support IPv6 addresses.

If there is a non-zero value in **wbep_client_address**, this value is used, and any value in **wbep_client_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **wbep_client_address** to allow the value in **wbep_client_ipv6_address** to be used.

wbep_client_ipv6_address
(Input only)

The colon hexadecimal IPv6 or dotted decimal IPv4 address of the client. This field can be up to 39 characters in length.

This field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **wbep_client_address** is not specified. This field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the client, or the IPv4 address of the client in IPv6 format. For more information on IP address format, see "IP addresses" on page 6.

wbep_client_address_len
(Input only)

The length of the dotted decimal IP address contained in **wbep_client_address**. This field contains zeros if the address is IPv6 format.

wbep_client_ipv6_address_len
(Input only)

The length of the IP address contained in **wbep_client_ipv6_address** or **wbep_client_address**.

wbep_close_conn
(Output only)

A 1-character field (Y or N) that indicates whether the connection is closed after the response is sent to the client. The default value of N indicates that the connection is not closed.

wbep_converter_program
(Input only)

The name of the converter program, if one is used, for the failing request.

wbep_converter_reason
(Input only)

The reason code returned by the converter program, if invoked.

wbep_converter_response
(Input only)

The response code returned by the converter program, if invoked.

wbep_error_code
(Input only)

The error code identifying the error detected.

wbep_eyecatcher
(Input only)

A character field containing an eyecatcher to help with diagnostics. DFHWBA sets this to `>wbepca` before calling the Web error program.

wbep_failing_program
(Input only)

An 8-character field containing the name of the program in which the error occurred.

wbep_http_response_code
(Input only)

The default HTTP status code returned by CICS for this error.

wbep_length
(Input only)

The length of the DFHWBEPC copybook.

wbep_message_len
(Input only)

The length of the CICS message text addressed by **wbep_message_ptr**.

wbep_message_number
(Input only)

A fullword number of the CICS WB domain message associated with the error.

wbep_message_ptr
(Input only)

A pointer to the CICS message text associated with this error.

wbep_response_len
(Input and output)

On input, this field is the fullword length of the default HTTP response for this error. CICS provides only a default response for HTTP requests; for non-HTTP requests, this field is zero. On output, this field contains the length of the default HTTP response, or of a modified response in the same block of storage, or of a replacement response in a new block of storage.

wbep_response_ptr
(Input and output)

On input, pointer points to a block of storage containing the default HTTP response for this error. CICS provides only a default response for HTTP requests. The default response is a complete HTTP response, including a status line, HTTP headers and message body. On output, this pointer either points to the same block of storage, containing the original or a modified version of the default response, or a pointer to a new block of storage containing a replacement response. If DFHWBEP successfully uses the **EXEC CICS** WEB SEND command to create a new response and sends it to the Web client, CICS ignores and discards the HTTP response in the block of storage. Otherwise, the response in the block of storage is sent to the Web client.

wbep_server_address
(Input only)

The 15-character IPv4 address of the server (CICS as an HTTP server) that must be set, if **wbep_server_ipv6_address** is not specified. **wbep_server_address** does not support IPv6 addresses.

If there is a non-zero value in **wbep_server_address**, this value is used, and any value in **wbep_server_ipv6_address** is ignored. Therefore, if you are using IPv6 addressing you must clear the contents of **wbep_server_address** to allow the value in **wbep_server_ipv6_address** to be used.

wbep_server_ipv6_address
(Input only)

A 16-byte field that must be set if you are using IPv6 addressing, or if you are using IPv4 addressing and **wbep_server_address** is not specified. This

field supports both IPv4 and IPv6 addresses and is set to the binary IPv6 address of the server (CICS as an HTTP server), or the IPv4 address of the server in IPv6 format. For more information on IP address format, see "IP addresses" on page 6.

wbep_server_address_len
(Input only)

The length of the dotted decimal IPv4 address contained in **wbep_server_address**. This field contains zeros if the address is IPv6 format.

wbep_server_ipv6_address_len
(Input only)

The length of the IP address contained in **wbep_server_ipv6_address** or **wbep_server_address**.

wbep_target_program
(Input only)

The target user-written application program that is designated to handle the Web client request.

wbep_tcpipservice_name
(Input only)

The name of the TCPIPSERVICE definition for the port on which the request is received.

wbep_version
(Input only)

A halfword binary number that indicates which version of the parameter list is currently being used. It is set using the constant value **wbep_current_version**.

wbep_suppress_abend
(Output only)

A 1-bit flag which, when set on, suppresses the abend AWBM.

Appendix I. The DFHWBCLI Web Client Interface

DFHWBCLI is a CICS-supplied utility program that you can invoke via **EXEC CICS** LINK to provide Web client services or outbound HTTP. It is supported in CICS Transaction Server for z/OS, Version 4 Release 1 for upgrade purposes.

The functions of the DFHWBCLI Web Client Interface are retained for compatibility reasons. To gain enhanced functionality, you can upgrade 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. "Making HTTP requests through CICS as an HTTP client" on page 148 describes how HTTP client requests can now be made.

If you want to use DFHWBCLI, you must set up CICS to use a name server. See Chapter 4, "Configuring CICS Web support base components," on page 53.

To use DFHWBCLI, you must link to it with a commarea that contains a parameter list whose contents are mapped by the following copybooks:

- DFHWBCLD for Assembler
- DFHWBCLO for Cobol
- DFHWBCLL for PL/I
- DFHWBCLH for C

The parameters have the following meanings:

WBCLI_VERSION_NO
a one-byte binary number that specifies the version number of this parameter list. It should be set to the value specified by the symbolic constant WBCLI_VERSION_CURRENT.

WBCLI_FUNCTION
A one-byte binary number that specifies the function that you want DFHWBCLI to execute. It should be set to one of the following values:

0 (WBCLI_FUNCTION_CONVERSE)
Send an HTTP request to a target server and receive the corresponding response

1 (WBCLI_FUNCTION_SEND)
Send an HTTP request to a target server and return control without waiting for the response

2 (WBCLI_FUNCTION_RECEIVE)
Wait for and receive the response to the HTTP request sent by a previous SEND function

3 (WBCLI_FUNCTION_INQUIRE_PROXY)
Request the name of the proxy server that was specified in the `INITPARM=(DFHWBCLI,'PROXY=http://....')` system initialization parameter

4 (WBCLI_FUNCTION_CLOSE)
Close the connection previously established by a SEND function, but without waiting for the HTTP response

WBCLI_METHOD
A one-byte binary number that specifies the HTTP method to be specified in the HTTP request. It should be set to one of the following values:

1 (WBCLI_METHOD_GET)

2 (WBCLI_METHOD_POST)

WBCLI_FLAGS
A one-byte binary bitstring that can be used to specify options associated with the HTTP request and its expected response. The bits in the bitstring may be set to the following values:

1... .... (WBCLI_OFFSET_MODE)
Pointer values in the parameter list are specified as offset values from the start of the parameter list. This implies that all the targets of such pointers are contained within the commarea.

.1.. .... (WBCLI_DOCUMENT)
The HTTP request body is a CICS document which was created by the DOCUMENT CREATE command and is specified by the document token in WBCLI_REQUEST_DOCTOKEN

..1. .... (WBCLI_USE_PROXY)
The HTTP request is to be sent via the proxy server whose URL is specified in WBCLI_PROXY_URL_PTR

...1 .... (WBCLI_SET_RESP_BUFFER)
CICS is to acquire a suitably sized buffer to contain the HTTP response body, and return its address in WBCLI_REQUEST_BODY_PTR

Note: This address is not made into an offset, regardless of the setting of WBCLI_OFFSET_MODE

.... ..1. (WBCLI_NATIVE_REQUEST_BODY)
The application will provide the HTTP request body in its native form, and CICS does not need to translate it from EBCDIC to ASCII

....1 (WBCLI_NATIVE_RESPONSE_BODY)
The application will handle the HTTP response body in its native form, and CICS does not need to translate it from ASCII to EBCDIC

WBCLI_RESPONSE
A halfword binary number that is set to one of the following values to indicate the outcome of the function:

0 (WBCLI_RESPONSE_OK)

4 (WBCLI_RESPONSE_EXCEPTION)

8 (WBCLI_RESPONSE_DISASTER)

WBCLI_REASON
A halfword binary number that is set to one of the following values to qualify the response code:

1 (WBCLI_REASON_INVALID_URL)
The format of the URL located by WBCLI_URL_PTR is invalid, or the host location cannot be resolved by the nameserver

2 (WBCLI_REASON_INVALID_HEADER)
One of the HTTP headers in the list located by WBCLI_HEADER_PTR is not in the correct format

3 (WBCLI_REASON_INVALID_DOCUMENT)
The document token specified in WBCLI_REQUEST_DOCTOKEN does not locate a valid CICS document

4 (WBCLI_REASON_GETMAIN_ERROR)
An error occurred while DFHWBCLI was attempting to obtain storage for one of its internal workareas

5 (WBCLI_REASON_PROXY_ERROR)
The proxy server located by WBCLI_PROXY_URL_PTR could not be found or returned an error response

6 (WBCLI_REASON_SOCKET_ERROR)
An unexpected response was returned when performing a socket operation

7 (WBCLI_REASON_HTTP_ERROR)
An unexpected HTTP response was returned by the server

8 (WBCLI_REASON_TRANSLATE_ERROR)
An error was returned while CICS was translating data between the host codepage and the server codepage. This could be because the required translation is not supported by CICS

9 (WBCLI_REASON_TRUNCATED)
The length of the user-provided response buffer specified in WBCLI_RESPONSE_BODY_LEN is insufficient to contain the response returned by the server. Data in excess of this length has been discarded.

10 (WBCLI_REASON_INVALID_HEADER_LENGTH)
The length of the user-provided request header specified in WBCLI_HEADER_LEN was invalid.

11 (WBCLI_REASON_INVALID_BODY_LENGTH)
The length of the user-provided request body specified in WBCLI_REQUEST_BODY_LEN was invalid.

WBCLI_SESSION_TOKEN
An opaque eight-byte binary token that represents the connection established with the HTTP server. It is set by the SEND function and is required by the RECEIVE and CLOSE functions. It is not used by the other functions.

WBCLI_URL_PTR
The address of an EBCDIC character string that contains the URL (Uniform Resource Locator) of the destination HTTP server. The URL must be fully qualified: that is, it must begin with '`http://`' or '`https://`'.

WBCLI_URL_LEN
A fullword binary number that contains the length of the URL located by WBCLI_URL_PTR.

WBCLI_PROXY_URL_PTR
The address of an EBCDIC character string that contains the URL (Uniform Resource Locator) of a proxy server that may be required to access remote sites outside your firewall. The URL must be fully qualified: that is, it must begin with '`http://`'. To use the proxy, you must also set the WBCLI_USE_PROXY flag.

WBCLI_PROXY_URL_LEN
A fullword binary number that contains the length of the URL located by WBCLI_PROXY_URL_PTR.

WBCLI_HEADER_PTR
The address of list of HTTP headers that are to be sent with the HTTP request. The headers must be encoded in EBCDIC, in the following form:

```
headername: headervalue§headername: headervalue§ ...
```

where
headername
is the name of the header
headervalue
is the value of the header

The colon (**:**) and space that separate these two should be present as shown; the '§' shown here should be replaced by one or more of the following delimiters:
carriage return (X'0D')
line feed (X'25')
new line (X'15')
field separator (X'1E')

Note: These delimiters are not used when the headers are sent: CICS uses the architecturally correct HTTP delimiters.
You may code as many headers in the list as you need. However, you must not include the following headers, as CICS will provide them:
Host
User-Agent
Content-Length
Content-Type

You do not need to provide a delimiter following the last header in the list.

WBCLI_HEADER_LEN
A fullword binary number that contains the length of the list of headers located by WBCLI_HEADER_PTR.

WBCLI_REQUEST_DOCTOKEN
A 16-byte binary document token, created by the DOCUMENT CREATE command, that represents a CICS document that is to be used as the HTTP request body. You must indicate that you are using this token by setting the WBCLI_DOCUMENT flag.

WBCLI_REQUEST_BODY_PTR
The address of an EBCDIC character string that contains the entire contents of the HTTP request body. This parameter is used when the WBCLI_DOCUMENT flag is *not* set.

WBCLI_REQUEST_BODY_LEN
A fullword binary number that contains the length of the request body that is located by WBCLI_REQUEST_BODY_PTR.

WBCLI_RESPONSE_BODY_PTR
The address of a buffer in which DFHWBCLI returns the HTTP response body from the server.

- If flag WBCLI_SET_RESP_BUFFER is not set, this address and WBCLI_RESPONSE_BODY_LEN must be set by the caller. If this buffer is not large enough to contain the response body, it is truncated.

- If flag WBCLI_SET_RESP_BUFFER is set, this address and WBCLI_RESPONSE_BODY_LEN are ignored. CICS obtains a new buffer which is large enough to contain the entire response, and its address is returned in this field. This address is never converted into an offset, whatever the value of the WBCLI_OFFSET_MODE flag.

Normally, CICS frees the storage at this address when the task that invokes DFHWBCLI ends. Alternatively, you can free the storage earlier by issuing an **EXEC CICS** FREEMAIN command in your application program. You are advised to do so when DFHWBCLI is called repeatedly in a long-running task, in order to prevent CICS going short-on-storage.

WBCLI_RESPONSE_BODY_LEN
A fullword binary number that contains the length of the response buffer located by WBCLI_RESPONSE_BODY_PTR.

- On input, if WBCLI_SET_RESP_BUFFER is not set, use this parameter to specify the length of the user-provided buffer.
- On output, it contains the actual length of the response body that was returned.

WBCLI_MEDIATYPE
A 40-byte EBCDIC blank-padded character string that contains the IANA media type (also known as the MIME type) of the HTTP body.

- On input, use this parameter to specify the media type of the HTTP request body. This media type will be sent in the HTTP Content-Type header
- On output, it will contain the media type of the HTTP response body, as received in the HTTP Content-Type header.

The media type must be specified for SEND requests that use the POST method (requests where WBCLI_FUNCTION_SEND and WBCLI_METHOD_POST are both set).

WBCLI_CHARSET
A 40-byte EBCDIC character string that contains the IANA character set of the HTTP body.

- On input, if WBCLI_NATIVE_REQUEST_BODY is not set, use this parameter to specify the name of the character set into which you want CICS to translate the HTTP request body. The character set you specify is used to qualify the media type in the HTTP Content-Type header. If you do not specify a value, the default of `iso-8859-1` is assumed only if WBCLI_MEDIATYPE includes the value TEXT.
- On output, it will contain the character set of the HTTP response body as received in the HTTP Content-Type header. This character set is used to translate the HTTP response body (unless the WBCLI_NATIVE_RESPONSE_BODY is set).

WBCLI_HOST_CODEPAGE
A 10-character EBCDIC blank-padded character string that contains the name of the EBCDIC code page used by your application. It is used in combination with WBCLI_CHARSET to determine what translation is to be performed on the HTTP document bodies (unless translation is suppressed by the WBCLI_NATIVE_REQUEST_BODY or WBCLI_NATIVE_RESPONSE_BODY flags). If it is omitted, CICS uses codepage 037.

WBCLI_HTTP_STATUS_CODE
A three-digit numeric EBCDIC character string in which the HTTP status code is returned. This indicates whether the HTTP request was successful or not. The 200 status code is used for a normal response, and other status codes in

the 2xx range also indicate success. Other status codes indicate that there is an error that prevents fulfilment of the request, or that the client needs to do something else in order to complete its request successfully, such as following a redirection URL.

Appendix J. Reference information for DFH$WBST and DFH$WBSR, state management samples

Two state management sample programs, DFH$WBST and DFH$WBSR, are supplied with CICS Web support. They allow a transaction to save data for later retrieval by the same transaction, or by another transaction. The saved data is accessed by a token that is created by the state management program for the first transaction. The first transaction must pass the token to the transaction that is to retrieve the data. DFH$WBST uses a GETMAIN command to allocate storage for the saved data. DFH$WBSR saves the data in temporary storage queues, one for each token, so that, with suitable temporary storage table definitions, the data can be accessed from several CICS systems. The rest of this section applies equally to either program.

The state management programs provide the following operations:
- Create a new token.
- Store information and associate it with a previously-created token.
- Retrieve information previously associated with a token.
- Destroy information associated with a token, and invalidate the token.

DFH$WBST also removes information and tokens that have expired. You can run this program periodically to purge state data which has expired:
- To purge all state data which has not been updated for one hour, run the program as transaction CWBT.
- To purge all state data, run the program as transaction CWBP.

The layout of the 268-byte COMMAREA is shown in the following table. You must clear the COMMAREA to binary zeros before setting the inputs for the function you require.

Table 28. Parameters for the state management program

Offset	Length	Type	Value	Notes
0	4	C		Eyecatcher
4	1	C	'C' 'R' 'S' 'D'	Create Retrieve Store Destroy This is the function code. It is a required input to every call.
5	1	X		Return code. This is an output from every call.
6	2	X		Reserved.
8	4	F		Token. This is an output from a Create call, and an input to every other call.
12	256	C		User data. This is an input to the Create and Store calls, and an output from a Retrieve call. It is not used in other calls.

The return codes are as follows:

0 The requested function was performed.

- If the function was Create, a new token is available at offset 8.
- If the function was Retrieve, the entity body associated with the input token at offset 8 is now in the entity body area at offset 12.
- If the function was Store, the input entity body at offset 12 is now associated with the input token and offset 8. Any entity body previously associated with the token is overwritten.
- If the function was Destroy, the data associated with the input token at offset 8 has been discarded, and the token is no longer valid.

2 The function code at offset 4 was not valid. Correct the program that sets up the COMMAREA.

3 The function was Create, but a GETMAIN command gave an error response.

4 The function was Retrieve, Store, or Destroy, but the input token at offset 8 was not found. Either the input token is not a token returned by Create, or it has expired.

5 A WRITEQ TS command gave an error response when writing internal data to a temporary storage queue.

7 An ASKTIME command gave an error response.

8 A READQ TS command gave an error response when reading internal data from a temporary storage queue.

9 An ASKTIME command gave an error response during time-out processing.

11 The function was Create, but a WRITEQ TS command gave an error response. This return code is produced only by DFH$WBSR.

12 The function was Retrieve, but a READQ TS command gave an error response. This return code is produced only by DFH$WBSR.

13 The function was Store, but a WRITEQ TS command gave an error response. This return code is produced only by DFH$WBSR.

14 The function was Destroy, but a DELETEQ TS command gave an error response. This return code is produced only by DFH$WBSR.

Appendix K. The CICS Web server plug-in

The functions of the CICS Web server plug-in are retained for compatibility reasons. You are recommended to migrate to solutions that make use of CICS Web services, CICS Web support, or the CICS Transaction Gateway.

This supplied plug-in enables a passthrough mechanism from the IBM HTTP Server through the external CICS interface (EXCI) and into CICS Web support, using the CICS business logic interface. The maximum amount of data that can be passed on this interface is 32 KB.

Configuring the IBM HTTP Server

The functions of the CICS Web server plug-in are retained for compatibility reasons. You are recommended to use the CICS Transaction Gateway in new applications.

About this task

You have to change the configuration information in the IBM HTTP Server if it is to use the CICS business logic interface to provide its service. *z/OS HTTP Server Planning, Installing, and Using*, SC34-4826, gives details of the configuration statements.

You can use the following procedure:

Procedure

1. You must set up CICS as follows:
   a. Initialize the CICS region with ISC=YES.
   b. Install the RDO group DFHWEB.
   c. Define a generic connection for EXCI; for example, by installing the sample group DFH$EXCI.
   d. Ensure that IRC is open.
2. Define the CICSTS41.CICS.SDFHDLL1 load library and CICSTS41.CICS.SDFHEXCI to RACF program control. RACF program control notes the volume serial number of the volume containing the library and does not allow the use of a different volume. If you later move the load library or the CICSTS41.CICS.SDFHEXCI library to another volume, you must redefine it to RACF Program Control.
3. Add the CICSTS41.CICS.SDFHDLL1 data set and the CICSTS41.CICS.SDFHEXCI library to the STEPLIB concatenation in the JCL for the IBM HTTP Server. SDFHEXCI and SDFHDLL1 are downwardly compatible with all supported releases of CICS.
4. Use the following command in the directory that contains the httpd.conf file for the IBM HTTP Server:

   ```
   ln -e DFHWBAPI dfhwbapi.so
   ```

   When it is used in the STEPLIB concatenation, this command establishes a link from the IBM HTTP Server's home directory to the DLL dfhwbapi.so in member DFHWBAPI in the CICSTS41.CICS.SDFHDLL1 library.

5. Add one or more service directives to the httpd.conf file. Service directives map the URL entered by the end user to the CICS resources that will satisfy the request. Service directives for DFHWBAPI have the following format:

   ```
   Service  /sourceurl/*  /home/dfhwbapi.so:DFHService/targeturl/*
   ```

   where the values are:

   home is the directory that contains the httpd.conf file for the IBM HTTP Server.

   sourceurl
   is a string of characters that selects an incoming URL to be processed by DFHWBAPI. The asterisk following it is a wildcard string representing the remaining characters of the incoming URL. *sourceurl* can be in any format, so details such as the *applid* and the *transaction* can be hidden from end users.

   targeturl
   targeturl is a string of characters that DFHWBAPI will use to determine which CICS resources will satisfy the user request. After substitution of the wildcard, *targeturl* must be in the format:

   ```
   /applid/converter/tran/program/filename
   ```

   where the values are:

   applid the application id of the target CICS region

   converter
   the name of the converter program to be used in the CICS region, or CICS if no converter is to be used.

   tran the transaction to be executed in the CICS region. Because the transaction is the target of an EXCI request, it should not be the Web alias transaction CWBA, but should be a mirror transaction, such as CSM3. The transaction receives *targeturl/**, not *sourceurl/**, as the incoming URL.

   program
   the name of the program to be executed in the CICS region.

   filename
   is any further information that will be examined by *program*.

   If DFHWBAPI is used to access 3270 applications, CICS generates HTML forms which are displayed on the Web client. The URL which CICS inserts in the HTML form matches the *targeturl* used in the previous request. To handle this situation, you must provide a service directive of the following form, in addition to those described above:

   ```
   Service  /targeturl/*  /home/dfhwbapi.so:DFHService
   ```

   In this case, the *targeturl* is passed unchanged to DFHWBAPI.

6. If you want to display the graphic files that are referenced from some of the CICS-supplied template definitions, include a directive as follows:

   ```
   Service  /dfhwbimg/*  /home/dfhwbapi.so:DFHService/applid/DFHWBIMG/CSM3/*
   ```

   where *applid* specifies the CICS region that will supply the graphics files (this might not be the same CICS region that does the bridge work). DFHWBIMG is a special-purpose CICS-supplied converter program used by the CICS Web bridge.

7. If you are accessing a CICS Web application using both CICS Web support and the CICS business logic interface, you must specify the same host code page for both. The default host code page for CICS is IBM-037, but for the IBM HTTP Server it is IBM-1047.
   - To change the default code page for the IBM HTTP Server, use the DefaultFsCp configuration directive. For example:

     ```
     DefaultFsCp IBM-1047
     ```
   - To change the default code page used by CICS, specify it in the DOCCODEPAGE system initialization parameter; for example, `DOCCODEPAGE=1047`.

     Documents and document fragments referenced using this default must be encoded in the specified code page. In particular, if you are using document templates generated from BMS map definitions, you must use a template customization macro to change the code page in which the templates are generated. Use the **CODEPAGE** parameter of the DFHMDX macro to specify this; for example:

     ```
     DFHMDX MAPSET=*,MAP=*,CODEPAGE=1047
     ```

     For more information on customizing templates generated from BMS map definitions, see Chapter 16, "Creating HTML templates from BMS definitions," on page 211.

What to do next

Escaped data and the IBM HTTP Server

If you use the IBM HTTP Server and CICS business logic interface to access the same CICS application program, you must make sure that escaped data is handled consistently in both cases.

The IBM HTTP Server passes data to the CICS application program in its unescaped form; therefore, you must ensure that CICS Web support does the same.

For more information, see "Selecting escaped or unescaped data from an analyzer program" on page 134

Processing examples for IBM HTTP Server

Figure 27 shows how the CICS Web support processes a request from a Web client that is connected to the IBM HTTP Server.

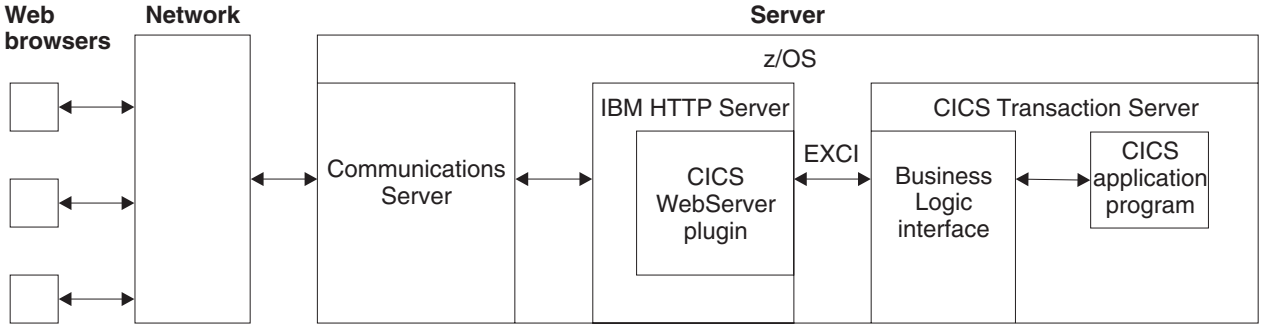


Figure 27. Processing a request from the IBM HTTP Server

1. The Web client constructs an HTTP request which is passed across the network to Communications Server.
2. Communications Server relays the request to IBM HTTP Server.
3. IBM HTTP Server calls the CICS Web server plug-in.
4. The CICS Web server plug-in constructs a request for the CICS business logic interface and passes it to CICS using the External CICS Interface (EXCI).
5. The CICS business logic interface invokes the requested CICS application program and returns any output in the COMMAREA.

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 want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Bibliography

CICS books for CICS Transaction Server for z/OS

General

CICS Transaction Server for z/OS Program Directory, GI13-0536
CICS Transaction Server for z/OS What's New, GC34-6994
CICS Transaction Server for z/OS Upgrading from CICS TS Version 2.3, GC34-6996
CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.1, GC34-6997
CICS Transaction Server for z/OS Upgrading from CICS TS Version 3.2, GC34-6998
CICS Transaction Server for z/OS Installation Guide, GC34-6995

Access to CICS

CICS Internet Guide, SC34-7021
CICS Web Services Guide, SC34-7020

Administration

CICS System Definition Guide, SC34-6999
CICS Customization Guide, SC34-7001
CICS Resource Definition Guide, SC34-7000
CICS Operations and Utilities Guide, SC34-7002
CICS RACF Security Guide, SC34-7003
CICS Supplied Transactions, SC34-7004

Programming

CICS Application Programming Guide, SC34-7022
CICS Application Programming Reference, SC34-7023
CICS System Programming Reference, SC34-7024
CICS Front End Programming Interface User's Guide, SC34-7027
CICS C++ OO Class Libraries, SC34-7026
CICS Distributed Transaction Programming Guide, SC34-7028
CICS Business Transaction Services, SC34-7029
Java Applications in CICS, SC34-7025

Diagnosis

CICS Problem Determination Guide, GC34-7034
CICS Performance Guide, SC34-7033
CICS Messages and Codes, SC34-7035
CICS Diagnosis Reference, GC34-7038
CICS Recovery and Restart Guide, SC34-7012
CICS Data Areas, GC34-7014
CICS Trace Entries, SC34-7013
CICS Supplementary Data Areas, GC34-7015
CICS Debugging Tools Interfaces Reference, GC34-7039

Communication

CICS Intercommunication Guide, SC34-7018
CICS External Interfaces Guide, SC34-7019

Databases

CICS DB2 Guide, SC34-7011
CICS IMS Database Control Guide, SC34-7016

CICS Shared Data Tables Guide, SC34-7017

CICSPlex SM books for CICS Transaction Server for z/OS

General

CICSPlex SM Concepts and Planning, SC34-7044
CICSPlex SM Web User Interface Guide, SC34-7045

Administration and Management

CICSPlex SM Administration, SC34-7005
CICSPlex SM Operations Views Reference, SC34-7006
CICSPlex SM Monitor Views Reference, SC34-7007
CICSPlex SM Managing Workloads, SC34-7008
CICSPlex SM Managing Resource Usage, SC34-7009
CICSPlex SM Managing Business Applications, SC34-7010

Programming

CICSPlex SM Application Programming Guide, SC34-7030
CICSPlex SM Application Programming Reference, SC34-7031

Diagnosis

CICSPlex SM Resource Tables Reference, SC34-7032
CICSPlex SM Messages and Codes, GC34-7035
CICSPlex SM Problem Determination, GC34-7037

Other CICS publications

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 4 Release 1.

Designing and Programming CICS Applications, SR23-9692

CICS Application Migration Aid Guide, SC33-0768

CICS Family: API Structure, SC33-1007

CICS Family: Client/Server Programming, SC33-1435

CICS Family: Interproduct Communication, SC34-6853

CICS Family: Communicating from CICS on System/390, SC34-6854

CICS Transaction Gateway for z/OS Administration, SC34-5528

CICS Family: General Information, GC33-0155

CICS 4.1 Sample Applications Guide, SC33-1173

CICS/ESA 3.3 XRF Guide , SC33-0661

Other IBM publications

The following publications contain information about related IBM products.

UNIX System Services

z/OS UNIX System Services User's Guide, SA22-7801
z/OS UNIX System Services Command Reference, SA22-7802
z/OS UNIX System Services Messages and Codes, SA22-7807
z/OS UNIX System Services Programming Tools, SA22-7805
z/OS UNIX System Services Programming: Assembler Callable, SA22-7803
z/OS UNIX System Services File System Interface Reference, SA22-7808
z/OS Using REXX and z/OS UNIX System Services, SA22-7806

z/OS V1R1 UNIX System Services Parallel Environment: MPI Programming and Subroutine Reference, SA22-7812

z/OS Communications Server

z/OS Communications Server: IP Configuration Reference, SC31-8776
z/OS Communications Server: IP Configuration Guide, SC31-8775
z/OS Communications Server: IP Migration, GC31-8773
z/OS Communications Server: IP CICS Sockets Guide, SC31-8807
z/OS Communications Server: IP Application Programming Interface Guide, SC31-8788
z/OS Communications Server: IP Programmer's Reference, SC31-8787
z/OS Communications Server: IP User's Guide and Commands, SC31-8780
z/OS Communications Server: Quick Reference, SX75-0124
z/OS Communications Server: IP Diagnosis, GC31-8782

IBM Redbooks

IBM Redbooks are developed and published by IBM's International Technical Support Organization. They explore integration, implementation and operation of realistic customer scenarios.

The following IBM Redbooks contain information related to material in this publication:

Accessing CICS Business Applications from the World Wide Web, SG24-4547
How to Secure the Internet Connection Server for MVS/ESA, SG324-4803
Revealed! Architecting Web Access to CICS, SG24-5466
Workload Management for Web Access to CICS, SG24-6118

Information on the Web

URLs are provided in this book with the caveat that their permanence cannot be guaranteed.

RFCs (Request for Comments)

RFC 1945, *Hypertext Transfer Protocol - HTTP/1.0* (HTTP/1.0 specification): http://www.ietf.org/rfc/rfc1945.txt
RFC 1867, *Form-based File Upload in HTML*: http://www.ietf.org/rfc/rfc1867.txt
RFC 2616, Hypertext Transfer Protocol - HTTP/1.1 (HTTP/1.1 specification): http://www.ietf.org/rfc/rfc2616.txt
RFC 2617, HTTP Authentication: Basic and Digest Access Authentication: http://www.ietf.org/rfc/rfc2617.txt
RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax: http://www.ietf.org/rfc/rfc2396.txt
RFC 3023, XML Media Types: http://www.ietf.org/rfc/rfc3023.txt

RFC documents are published by the Internet Society and the Internet Engineering Task Force (IETF).

IANA (Internet Assigned Numbers Authority)

IANA-registered media types: http://www.iana.org/assignments/media-types/
IANA-registered character set names: http://www.iana.org/assignments/character-sets
IANA-registered port numbers: http://www.iana.org/assignments/port-numbers

World Wide Web Consortium (W3C)

W3C home page: http://www.w3.org/
W3C overview of HTTP: http://www.w3.org/Protocols/Overview.html
W3C HTML Home Page: http://www.w3.org/MarkUp/

IBM IBM developerWorks: http://www.ibm.com/developerworks/

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.

If you use the Web Terminal Translation Application (DFHWBTTA) to access CICS-supplied transactions, the accessibility features are those which your Web browser provides.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

V

W

X

Z

Readers' Comments — We'd Like to Hear from You

CICS Transaction Server for z/OS
Version 4 Release 1
Internet Guide

Publication No. SC34-7021-03

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: +44 1962 816151
- Send your comments via email to: idrcf@uk.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-7021-03

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM United Kingdom Limited
User Technologies Department (MP095)
Hursley Park
Winchester
Hampshire
United Kingdom
SO21 2JN

Fold and Tape | **Please do not staple** | Fold and Tape

SC34-7021-03

Cut or Fold
Along Line

SC34-7021-03