

CICS[®] Transaction Server for VSE/ESA[™]



Internet Guide

Release 1

CICS[®] Transaction Server for VSE/ESA[™]



Internet Guide

Release 1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 151.

First Edition (September 2000)

This edition applies to Release 1 of the IBM licensed program CICS Transaction Server for VSE/ESA, program number 5648-054, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable IBM system bibliography for current information on this product.

Order publications through your IBM representative or IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of the publication is a page entitled "Sending your comments to IBM". If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1994, 2000. All rights reserved.

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

Contents

Preface	ix
What this book is about	ix
How to use this book	ix
What you need to know to understand this book	ix
Notes on terminology	ix

Part 1. Overview 1

Chapter 1. Introduction	3
General concepts	6
Distributed computing	6
Security support	7
TCP/IP protocols	7
TCP/IP internet addresses and ports	8
Programming models	9
Comparing mechanisms	10
Accessing CICS from the Web	10
CICS and Java	11
CICS TRANSACTION GATEWAY	11
Application design	11
Separating business and presentation logic	11
Chapter 2. How this book is organized	13

Part 2. CICS Web support 15

Chapter 3. Introduction to CICS Web support	17
Types of requester	17
Types of service	17
Processing examples	18
Control flow in request processing	18
Using CICS Web support to call a program	18
Using CICS Web support to run a terminal-oriented transaction	20
Data flow in request processing	21
Using the CICS Web support commarea method to call a program	21
Chapter 4. Planning for CICS Web support	25
Defining the prerequisites for using CICS Web support	26
VSE/ESA	26
CICS	26
TCP/IP for VSE/ESA	26
URL format	26
Operations tasks	27
Chapter 5. Configuring CICS Web support	29
Controlling web support with system initialization parameters	29
Defining resources to CICS	29
CICS supplied resource definitions	30
DOCTEMPLATE definitions	30
TCPIPSERVICE definitions	31
TRANSACTION definitions for extra alias transactions	32
PROGRAM definitions for user-replaceable programs	33

Setting up a sub-library for the template manager	33
Defining a conversion table	33
Configuring TCP/IP for VSE/ESA	35
Reserving ports for CICS Web support	35
Identifying the TCP/IP server	35
Specifying a name server	35
Enabling lightpen support	36
Running the sample application	36
Chapter 6. Writing an analyzer for CICS Web support	37
Inputs	37
Outputs	38
Processing	38
Code page considerations for Web API applications	39
Code page considerations for Web commarea applications	39
Performance considerations	40
The default analyzer	40
Chapter 7. Writing a converter	43
Performance considerations	43
Writing a converter—Decode	44
Inputs	44
Outputs	44
Processing	44
Writing a converter—Encode	45
Inputs	45
Outputs	45
Processing	46
Chapter 8. The Web error program	47
The Web error program — general	47
Inputs	47
Outputs	48
Processing	48
Chapter 9. 3270 applications on the Web	49
Input to DFHWBTTA	49
Customizing the input to DFHWBTTA	51
Output from DFHWBTTA	51
Customizing the output from DFHWBTTA	52
Required contents for a heading template	52
Required contents for a footing template	53
Customizing with Encode	54
Lightpen operation	54
Chapter 10. Creating HTML templates from BMS definitions	57
Generating a standard template	57
Why customize the generation of templates?	57
Customization facilities	58
How to produce the HTML templates	58
Restricting the size of HTML templates	58
Writing a customizing macro definition	59
Customization examples	59
HTML and browser considerations	62
Limitations	62
The DFHMDX macro	63

The DFHWBOUT macro	66
Chapter 11. Writing CICS programs to process HTTP requests	69
Using HTTP requests	69
How to receive an HTTP request	70
Explaining HTTP responses	71
How to send an HTTP response	72
Handling Escaped Data.	73
Handling escaped data in commarea applications	73
Describing symbols, symbol table, and symbol list	74
Symbols in an HTML template	74
Symbol lists	74
Operational example.	75
Using the output of the environment variables program	76
Sample application programs.	76
Chapter 12. Displaying a template on a Web browser	77
How to display a template on a Web browser	77
Default CICS URL format	79
Chapter 13. Security for CICS Web support	81
Security for the HTML template manager sub-library	81
Security for CICS Web support transactions	81
Security for the alias	81
Using sample programs for security	82
The security sample programs	82
Using basic authentication sample programs	83
Chapter 14. Problem determination	85
Recovery procedures (CICS Web support).	86
Product design considerations (CICS Web support)	86
Troubleshooting	86
Defining the problem.	86
Documentation about the problem	87
Using messages and codes	87
CICS Web support and CICS business logic interface trace information	87
Numeric values of symbolic codes.	88
Dump and trace formatting	88
Debugging the user-replaceable programs.	89
Using EDF	89
Using trace entries	89
Writing messages	89
Trapping abends	89

Part 3. The CICS business logic interface 91

Chapter 15. Introduction to the CICS business logic interface	93
Types of requester	93
Processing examples	93
Controlling flow in request processing	94
Using the CICS business logic interface to call a program	94
Using the CICS business logic interface to run a terminal-oriented transaction	95
Passing data in request processing	96
Using the CICS business logic interface to call a program	96
Requesting a terminal-oriented transaction.	96

Chapter 16. Configuring the CICS business logic interface	101
Chapter 17. Programming tasks for client systems	103

Part 4. Appendixes 105

Appendix A. Reference information for DFHWBBLI	107
Business logic interface	108
Appendix B. Reference information for DFHWBADX	115
Summary of parameters	115
Function	115
Parameters	116
Responses and reason codes	118
DFHWBADX responses and reason codes	119
Appendix C. Reference information for the converter	121
Decode	122
Encode	127
Appendix D. Reference information for DFHWBTL	131
Parameters in the communication area	132
Responses and reason codes	134
Appendix E. Reference information for DFHWBENV	137
Appendix F. Reference information for DFH\$WBST and DFH\$WBSR	141
Appendix G. Reference information for DFHWBPA	143
Appendix H. Reference information for DFHWBEP	145
Parameters	145
Appendix I. HTML-coded character sets	149
Notices	151
Programming interface information	152
Trademarks and service marks	152
Bibliography	155
CICS Transaction Server for VSE/ESA Release 1 Library	155
Books from VSE/ESA 2.5 base program libraries	157
VSE/ESA Version 2 Release 5.	157
High-Level Assembler Language (HLASM)	157
Language Environment for VSE/ESA (LE/VSE)	158
VSE/ICCF	158
VSE/POWER	158
VSE/VSAM	158
VTAM for VSE/ESA.	159
Books from VSE/ESA 2.5 optional program libraries	161
C for VSE/ESA (C/VSE)	161
COBOL for VSE/ESA (COBOL/VSE)	161
DB2 Server for VSE	162
DL/I VSE	162

PL/I for VSE/ESA (PL/I VSE)	162
Screen Definition Facility II (SDF II)	163
TCP/IP for VSE/ESA	163
Information on the World Wide Web	165
HTTP/1.0	165
HTML	165
Determining if a publication is current	167
Index	169
Sending your comments to IBM	175

Preface

What this book is about

This book describes how you can make the CICS® transaction processing services of CICS Transaction Server for VSE/ESA® available to a variety of Internet users and TCP/IP-based applications.

How to use this book

This book is intended to show what CICS facilities are available to enable you to use your CICS system as a non-SNA server. Read “Part 1. Overview” on page 1 for general information, and for guidance about which other parts of the book to consult.

What you need to know to understand this book

Within this book, reference is made to Release 1. The function described in this book has been added since the release of CICS Transaction Server for VSE/ESA Release 1 and can be identified within the code as 1.1.1.

This book assumes that you are familiar with CICS, either as a system administrator or as a system or application programmer. Some parts of the book assume additional knowledge about CICS and other products.

Notes on terminology

When the term “CICS” is used without any qualification in this book, it refers to the CICS element of IBM® CICS Transaction Server for VSE/ESA.

In this release, the CICS Web interface is split into the Listener support for TCPIPSERVICE, and the protocol support for HTTP. This book now refers to the HTTP protocol support as “**CICS Web support**”. Within the product code, the term “**CICS Web interface**” remains synonymous with “**CICS Web support**”.

In this release, there are two ways of coding Web application programs.

1. **Commarea**-style applications are those that take a communication area containing an HTTP request as input, and build an HTTP response in the communication area.
2. **Web API** applications use the new WEB and DOCUMENT application programming interface to process the inbound HTTP request and build the response

Part 1. Overview

This part of the book outlines some of the ways in which you can make CICS transaction processing services available to a variety of Internet users.

This part contains:

- “Chapter 1. Introduction” on page 3
- “Chapter 2. How this book is organized” on page 13

Overview

Chapter 1. Introduction

This book describes the following sources of external requests, and the routes that they can use into CICS:

Web browsers

Web browsers can use a variety of methods:

CICS Web support

CICS Web support is a CICS facility for supporting Web browsers.

CICS Transaction Gateway

This is a workstation application that can accept requests from Web browsers and route them into CICS. It uses a CICS client and the EPI.

Java-enabled Web browsers

Java-enabled Web browsers can use applets to communicate with CICS. The applets can use CICS-provided Java classes to construct external call interface (ECI) and external presentation interface (EPI) requests. The Web browsers communicate with Web servers, and with one of the following:

CICS Transaction Gateway

This is a workstation application that uses a CICS client to route ECI and EPI requests to a CICS server.

The following types of external requests are described in other books:

3270 users

Users of the IBM 3270 Display System can start transactions. This is the most familiar method of introducing work to CICS Transaction Server for VSE/ESA.

User socket applications

User socket applications can use the CICS Sockets feature of TCP/IP for VSE/ESA. See the TCP/IP for VSE/ESA Programmer's Reference.

VSE™ applications

Applications running in VSE partitions can use the External CICS Interface (EXCI) to access CICS programs. See the *CICS External Interfaces Guide*.

CICS client applications

CICS client applications use a CICS client and the ECI or the EPI. See the *CICS Family: Client/Server Programming*.

Telnet clients

Telnet clients can use TN3270 to start transactions. See the TCP/IP for VSE/ESA Installation Guide.

CICS programs

Programs running in CICS servers on any platform can use EXEC CICS LINK to call a CICS program, or transaction routing to send transaction requests to CICS Transaction Server for VSE/ESA. Programs running in CICS Transaction Server for VSE/ESA can use the CICS front end programming interface (FEPI) to start transactions in the same or another instance of CICS Transaction Server for VSE/ESA. See the *CICS Front End Programming Interface User's Guide*.

Figure 2 on page 5 shows the principal ways of using CICS transaction processing services from outside CICS.

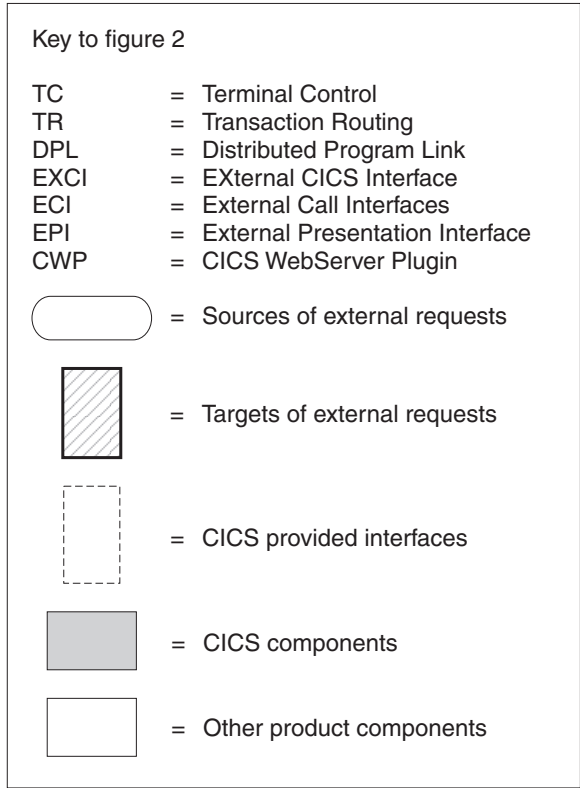


Figure 1.

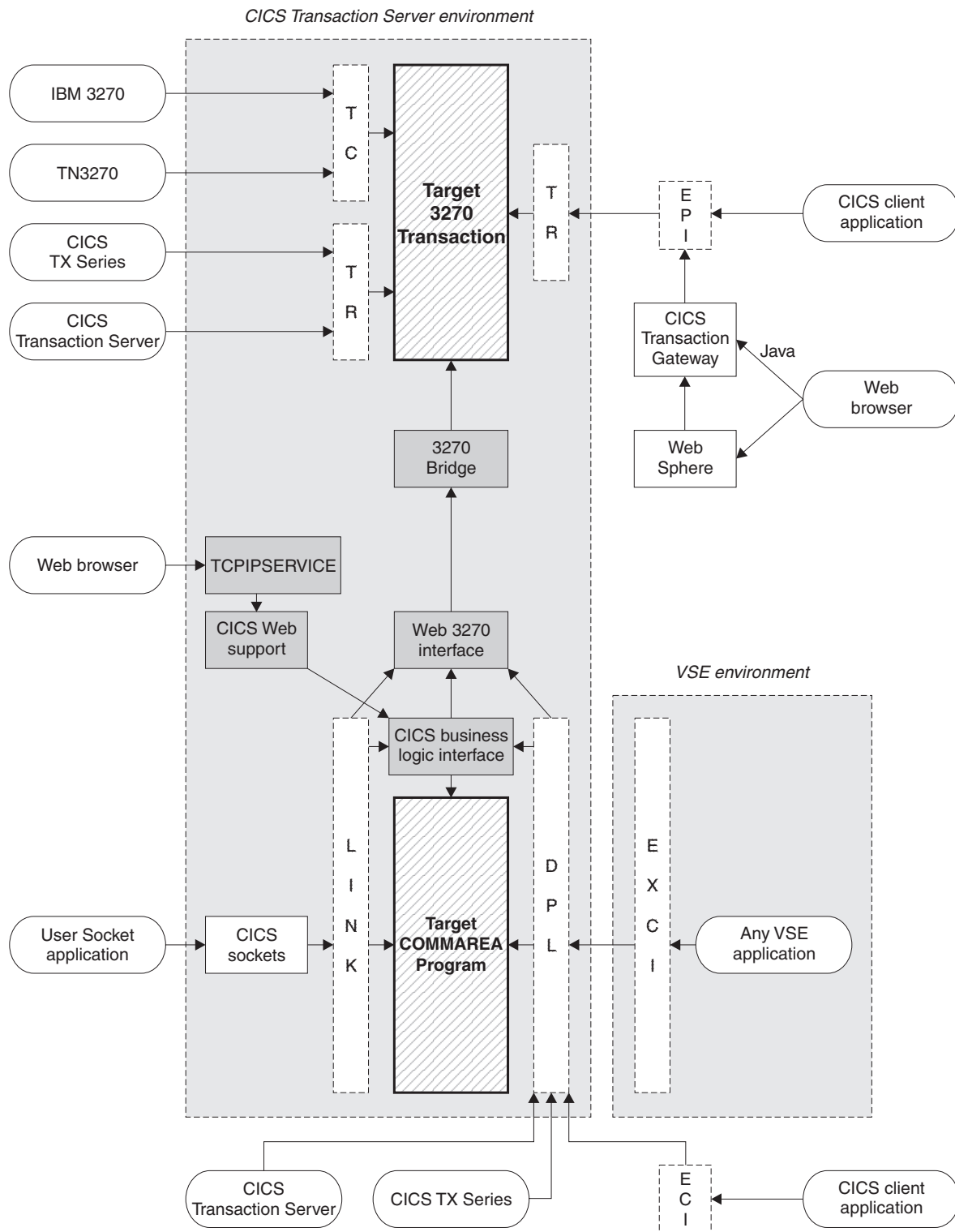


Figure 2. Client access to existing business logic

General concepts

All the mechanisms described in this book follow a similar pattern. A client is the source of the external request which comes into CICS over a network using a variety of transport protocols, or from another CICS region, using Inter Region Communication (IRC). CICS (or another product) provides a transport-specific listener (a long-running task) that starts another task (a facilitator such as an **alias** or a **mirror**), to process the incoming request. The facilitator uses CICS services to access the application.

The priorities of different alias transactions can be adjusted to determine the service that a client request receives. There must be enough free tasks to service the alias transactions as they are started by the listener. The CICS programs that service the client requests are subject to contention for resources in the CICS system, and to transmission delays if they are remote from the CICS system, or if they request the use of remote resources by function shipping or distributed program link.

The CICS server is independent of the application model (2/3-tier, 2/3 platforms). The listener/facilitator deals with the different transports used and sets the rules for which programming models are supported.

Distributed computing

Distributed computing involves the cooperation of two or more machines communicating over a network. The machines participating in the system can range from personal computers to super computers; the network can connect machines in one building or on different continents.

The main benefit of distributed computing is that it enables you to optimize your computing resources for both responsiveness and economy. For example, it enables you to:

- Share the cost of expensive resources, such as a typesetting and printing service, across many desktops. It also gives you the flexibility to change the desktop-to-server ratio, depending on the demand for the service.
- Allocate an application's presentation, business, and data logic appropriately. Often, the desktop is the best place to perform the presentation logic, as it is nearest to the end user and can provide highly responsive processing for such actions as drag and drop GUI interfaces.

Conversely, you may feel that the best place for the database access logic is close to the actual storage device - that is, on an enterprise or departmental server. The most appropriate place for the business logic may be less clear, but there is much to be said for placing this too in the same node as the data logic, thus allowing a single desktop request to initiate a substantial piece of server work without intervening network traffic.

Distributed computing enables you to make such trade-offs in a flexible way.

Along with the advantages of distributed computing come new challenges. Examples include keeping multiple copies of data consistent, keeping clocks in individual machines synchronized, and providing network-wide security. A system that provides distributed computing support must address these new issues.

CICS supports distributed computing and the client/server model by means of:

Distributed program link (DPL)

A CICS client program passes parameters to a remote CICS server program

and waits for the server to send data in reply. Parameters and data are exchanged by means of a communications area.

The external CICS interface (EXCI)

A VSE client program links to a CICS server program. This is similar to DPL.

The external call interface (ECI)

The ECI enables CICS Transaction Server for VSE/ESA server programs to be called from client programs running on a variety of operating systems. For information about CICS Clients, see the *CICS Family: Client/Server Programming* manual.

Function shipping

The parameters for a single CICS API request are intercepted by CICS code and sent from the client system to the server. The CICS mirror transaction in the server executes the request, and returns any reply data to the client program. This can be viewed as a specialized form of remote procedure call.

Asynchronous transaction processing

A CICS client transaction uses the EXEC CICS START command to initiate another CICS transaction, and pass data to it. The START request can be intercepted by CICS code, and function shipped to a server system. The client transaction and started transactions execute independently. This is similar to a remote procedure call with no response data.

Distributed transaction processing

A program in the client system establishes a conversation with a complementary program in the server, and exchanges messages. The programs may use the APPC protocols.

Transaction routing

Terminals owned by one CICS system to run transactions owned by another.

The CICS family of products runs on a variety of operating systems, and provides a standard set of functions to enable members to communicate with each other. For information about the CICS family, see the *CICS Family: Interproduct Communication* manual.

Security support

CICS Transaction Server for VSE/ESA supports:

- A single network signon (through the ATTACHSEC option of the DEFINE CONNECTION command)
- Authentication of the client system through bind-time security.

An external security manager provides resource access control and login facility.

In all the above scenarios the client environment must know which server CICS system to communicate with. This is normally done by specifying the name of the required remote CICS system in the definition of the relevant remote CICS resource, or in the client application program.

TCP/IP protocols

TCP/IP is a communication protocol used between physically separated computer systems. TCP/IP can be implemented on a wide variety of physical networks.

TCP/IP is a large family of protocols that is named after its two most important members, Transmission Control Protocol and Interface Protocol. Figure 3 on page 8

shows the TCP/IP protocols used in terms of the layered Open Systems Interconnection (OSI) model. For CICS users, who may be more accustomed to SNA, the left side of Figure 3 shows the SNA layers that correspond very roughly to the OSI layers.

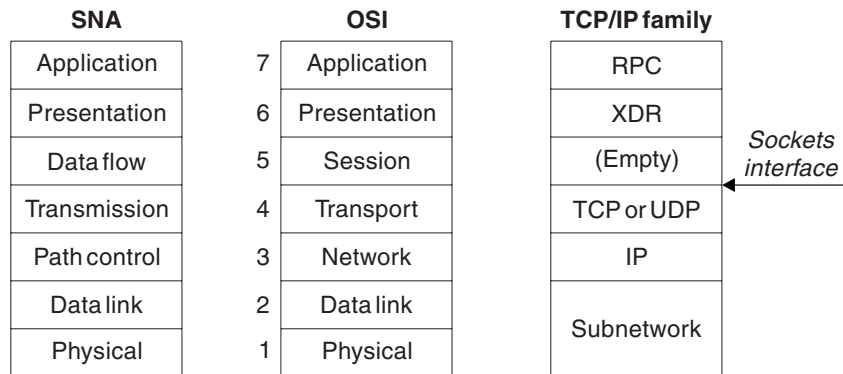


Figure 3. TCP/IP protocols compared to the OSI and SNA models

The protocols used by TCP/IP are shown in the right-hand box in Figure 3.

Internet Protocol (IP)

In terms of the OSI model, IP is a network-layer protocol. It provides a *connectionless* data transmission service, and supports both TCP and UDP. 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)

In terms of the OSI model, TCP is a transport-layer protocol. It provides a *connection-oriented* data transmission service between applications, that is, a connection is established before data transmission begins. TCP has more error checking than UDP.

User Datagram Protocol (UDP)

UDP is also a transport-layer protocol and is an alternative to TCP. It provides a connectionless data transmission service between applications. UDP has less error checking than TCP. If UDP users want to be able to respond to errors, the communicating programs must establish their own protocol for error handling. With high-quality transmission networks, UDP errors are of little concern.

ONC RPC and XDR

XDR and ONC RPC correspond to the sixth and seventh OSI layers.

Sockets interface

The interface between the fourth and higher layers is the *sockets* interface. In some TCP/IP implementations, the sockets interface is the API that customers use to write their higher-level applications.

TCP/IP internet addresses and ports

TCP/IP provides for process-to-process communication, which means that calls need an addressing scheme that specifies both the physical host connection (Host A and Host B in Figure 4 on page 9) and the software process or application (C, D, E, F, G, and H). The way this is done in TCP/IP is for calls to specify the host by an *internet address* and the process by a *port number*. You may find internet

addresses also referred to elsewhere as internet protocol (IP) addresses or host IDs.

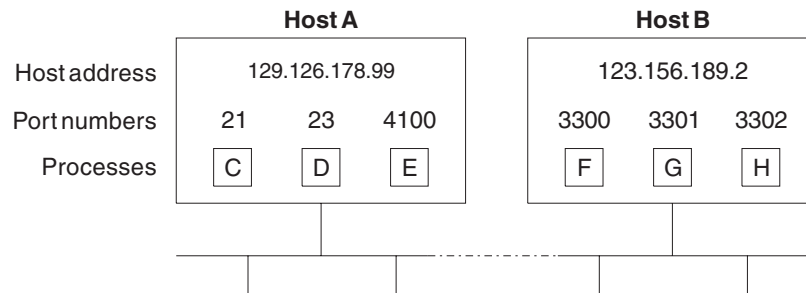


Figure 4. How applications are addressed

Internet addresses

Each host on a TCP/IP internet is identified by its internet address. An internet address is 32 bits, but it is usually displayed in dotted decimal notation. Each byte is converted to a decimal number in the range 0 to 255, and the four numbers are separated by dots thus: 129.126.178.99.

Remember that an internet is a collection of networks — hence the internet address must specify both the network and the individual host. How this is done varies with the size of the network. For example, in Figure 4, 129.126 could specify the network, and 178.99 could specify the host on that network.

Port numbers (for servers)

An incoming connection request specifies the server that it wants by specifying the server's port number. For instance, in Figure 4, a call requesting port number 21 on host A is directed to process C.

Well-known ports identify servers that carry standard services such as the File Transfer Protocol (FTP) or Telnet. The same service is always allocated the same port number, so, for example, FTP is always 21 and Telnet always 23. Networks generally reserve port numbers 1 through 255 for well-known ports.

Port numbers (for clients)

Client applications must also identify themselves with port numbers so that server applications can distinguish different connection requests. The method of allocating client port numbers must ensure that the numbers are unique; such port numbers are termed *ephemeral port numbers*. For example, in Figure 4, process F is shown with port number 3300 on host B allocated.

Programming models

The programming models implemented in CICS are inherited from those designed for 3270s, and exhibit many of the characteristics of conversational, terminal-oriented applications. There are basically three styles of programming model:

- Terminal-initiated, that is, the conversational model
- Distributed program link, that is, the RPC model
- START, that is, the queuing model.

Once initiated, the applications typically use these and other methods of continuing and distributing themselves, for example, with pseudoconversations, RETURN

IMMEDIATE or DTP. The main difference between these models is in the way that they maintain state (for example, security), and hence state becomes an integral part of the application design. This presents the biggest problem when you attempt to convert to another application model.

A pseudoconversational model is mostly associated with terminal-initiated transactions and was developed as an efficient implementation of the conversational model. With increased use of 1-in and 1-out protocols such as HTTP, it is becoming necessary to add the pseudoconversational characteristic to the RPC model.

State management and its associated token management, which were previously controlled by the terminal, now need additional techniques to support this move. Similarly, when START requests are disassociated from the terminal, difficulties arise in returning the requests to their starting point.

Comparing mechanisms

This topic compares accessing CICS from the Web, and using CICS with Java. It lists some of the characteristics and benefits of each interface. Your decision about which access mechanism to use depends on the type of client (for example, Web browser, Java). This affects the transport and presentation protocol that you use.

Accessing CICS from the Web

CICS Web support allows you to use a Web browser as a graphical user interface for business logic applications¹. Its main purpose is to allow you to build CICS HTML application utilities; it is not designed to perform as a full Web server. You should use a separate Web server for facilities such as:

- supplying GIFs, applets, and other items referenced from the CICS pages
- supporting News, e-mail, FTP, and Gopher daemons
- providing the proxy, firewall, and gateway services needed when connecting to the Internet.

Here are some of the things you should consider when choosing a CICS Web solution:

- The programming model you intend to use. For example, whether the target program is a commarea program or a 3270 transaction (BMS or non-BMS).
- How your applications are designed. Do you want a 2–tier solution, where a Web browser talks directly to CICS Web support by means of a Web server within VSE/ESA, or a 3–tier solution, where the Web server is external to VSE/ESA (for example, on AIX).
- Whether your application is contained entirely within CICS, or is a program outside CICS which needs access to CICS as part of a larger application.

If your program is entirely within CICS, you should consider using the CICS business logic interface. This way, you can use different front ends to existing programs without the need for the new client to understand the format of the commarea, or for the program to be aware of the different callers. Because you can use a converter, the format can be hidden and maintained in one place, and changes either to the client or to the program require changes only to the

1. A business logic application is one that communicates via a communications area, and from which all presentation logic has been removed. See “Application design” on page 11 for further details.

converter. The converter is then responsible for managing the translation of formats, a different one being specified on the CICS business logic interface depending on the caller.

- Whether the application is Web-aware. A Web-aware application understands HTTP and produces HTML without the need for a converter. “Chapter 11. Writing CICS programs to process HTTP requests” on page 69 describes two methods of writing Web-aware applications:
 - Web API applications, which use the EXEC CICS WEB and EXEC CICS DOCUMENT application programming interface to process the inbound HTTP request and build the response. This is the recommended method.
 - Commarea-style applications, which accept as input a communication area containing an HTTP request, and also build the HTTP response in the communication area. This method is retained for compatibility with previous releases.

CICS and Java

CICS supports one Java environment;

- Java support provided by the CICS Transaction Gateway.

CICS TRANSACTION GATEWAY

The Java language can be used to construct Java applets and Java applications, both of which are used in the CICS Transaction Gateway. Here, the Java executes outside the CICS environment, and access into CICS is provided by the Java classes supplied by the gateway. For example, an applet written for the Java gateway would use the `com.ibm.ctg.client.ECIRequest` class to produce an ECI call to communicate with a COBOL program using a commarea.

Application design

You can access existing applications originally designed for other environments, such as the Web use of the bridging facilities described in “Using CICS Web support to run a terminal-oriented transaction” on page 20, or write new ones specifically for a new environment. In general, it is good practice to split applications into a part containing the business code that is reusable, and a part responsible for presentation to the client. This technique enables you to improve performance by optimizing the parts separately, and allows you to reuse your business logic with different forms of presentation.

When separating the business and presentation logic, you need to consider the following:

- Avoid affinities between the two parts of the application.
- Be aware of the DPL-restricted API; see the *CICS Application Programming Reference* for details.
- Be aware of hidden presentation dependencies, such as EIBTRMID usage.

Separating business and presentation logic

Figure 5 on page 12 illustrates a simple CICS application that accepts data from an end user, updates a record in a file, and sends a response back to the end user. The transaction that runs this program is the second in a pseudoconversation. The first transaction has sent a BMS map to the end user’s terminal, and the second

Chapter 2. How this book is organized

Having read “Chapter 1. Introduction” on page 3 to get an understanding of the different ways of introducing work into CICS, use the rest of the manual as reference material. It is organized as follows:

- “Part 2. CICS Web support” on page 15 describes support for web browsers through CICS Web support.
- “Part 3. The CICS business logic interface” on page 91 describes the CICS business logic interface

Part 2. CICS Web support

This part of the book describes CICS Web support.

It contains:

- “Chapter 3. Introduction to CICS Web support” on page 17
- “Chapter 4. Planning for CICS Web support” on page 25
- “Chapter 5. Configuring CICS Web support” on page 29
- “Chapter 6. Writing an analyzer for CICS Web support” on page 37
- “Chapter 7. Writing a converter” on page 43
- “Chapter 8. The Web error program” on page 47
- “Chapter 9. 3270 applications on the Web” on page 49
- “Chapter 10. Creating HTML templates from BMS definitions” on page 57
- “Chapter 11. Writing CICS programs to process HTTP requests” on page 69
- “Chapter 12. Displaying a template on a Web browser” on page 77
- “Chapter 13. Security for CICS Web support” on page 81
- “Chapter 14. Problem determination” on page 85

CICS Web support

Chapter 3. Introduction to CICS Web support

This part of the book describes CICS Web support, a function of CICS that promotes access to CICS transaction processing services from outside CICS. It is primarily, though not exclusively, concerned with access from Web browsers on the Internet, or on an enterprise's intranet.

The CICS business logic interface is a callable program that allows a variety of callers to access the same Web-aware business logic as CICS Web support, but via a CICS link rather than via the CICS HTTP listener.

CICS Web support and the CICS business logic interface support the separation of presentation logic from business logic in application design. They also support the conversion of output that uses existing presentation methods, such as CICS basic mapping support (BMS), into others, particularly hypertext markup language (HTML). There is a brief discussion about the distinction between presentation logic and business logic in "Separating business and presentation logic" on page 11.

The rest of this chapter presents an overview of this facility. It contains the following sections:

- "Types of requester"
- "Types of service"
- "Processing examples" on page 18
- "Control flow in request processing" on page 18
- "Dealing with non-HTTP requests" on page 20
- "Data flow in request processing" on page 21

"Chapter 4. Planning for CICS Web support" on page 25 presents a list of tasks associated with planning, installing, customizing, programming, and operating the facilities.

Types of requester

The CICS Web support can deal with requests from these types of requester:

- Web browsers that are connected to a TCP/IP port that is reserved for the CICS Web support. A user-replaceable program relates the hypertext transfer protocol (HTTP) request to the required CICS transaction processing services.
- Non-HTTP clients — see "Dealing with non-HTTP requests" on page 20.
- Web browsers connected to an HTTP server that invokes the CICS business logic interface. See "Chapter 15. Introduction to the CICS business logic interface" on page 93. For example, the CICS Transaction Gateway uses this interface.

Types of service

CICS Web support supplies CICS transaction processing services in the following ways:

1. Using a non-terminal transaction to run a CICS program. A user-replaceable program maps data in the request to the communication area that the program is expecting. The user-replaceable program also maps the output communication area into the response format expected by the requester. If the CICS program is written to accept and process HTTP and HTML, the

user-replaceable program might not be needed. CICS provides support for manipulating HTML pages when the requester's protocol includes HTML.

2. Starting a CICS terminal-oriented transaction. This service is designed to be used when the request is an HTTP request, and contains HTML. CICS recognizes that this is a request for a terminal-oriented transaction from the format of the HTTP request. CICS provides a procedure and supporting tools for mapping 3270 data streams, including those produced by BMS maps, into HTML, and HTML into BMS. The user can customize this mapping, either by creating a macro definition, or by providing a user-replaceable program, or both. This is designed to allow existing 3270-based applications to function without modification.

Processing examples

Figure 7 shows how CICS Web support processes a request from a Web browser that is connected to a TCP/IP for VSE/ESA server.

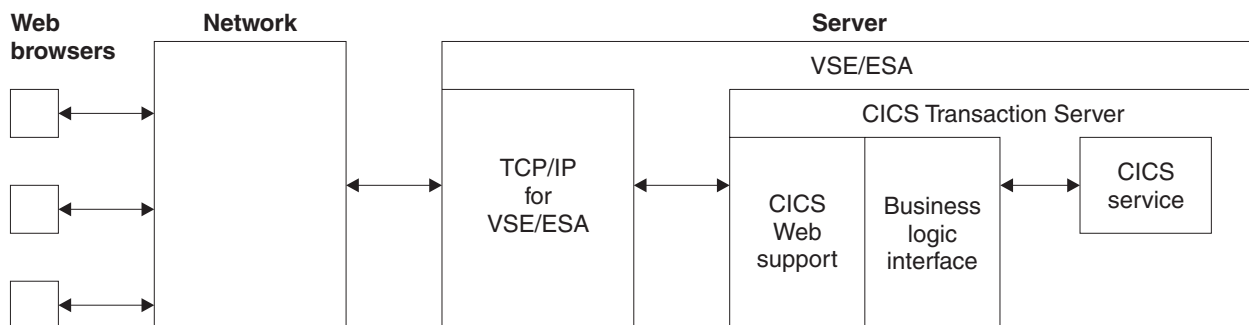


Figure 7. Processing a request to CICS Web support

The Web browser is an HTTP client. It constructs an HTTP request, which is passed across the network to TCP/IP for VSE/ESA in the server. TCP/IP for VSE/ESA relays the request to CICS Web support, which provides the requested service. The output is sent back to the Web browser in an HTTP response.

Control flow in request processing

To make decisions about the facilities to use, and how to customize them, you need to understand how CICS Web support interacts with the CICS business logic interface.

Using CICS Web support to call a program

Figure 8 on page 19 shows the control flow through CICS Web support to a CICS program. This would be the result of receiving a typical URL of the form:

```
http://ip_address:port/converter_program/cwba/user_program
```

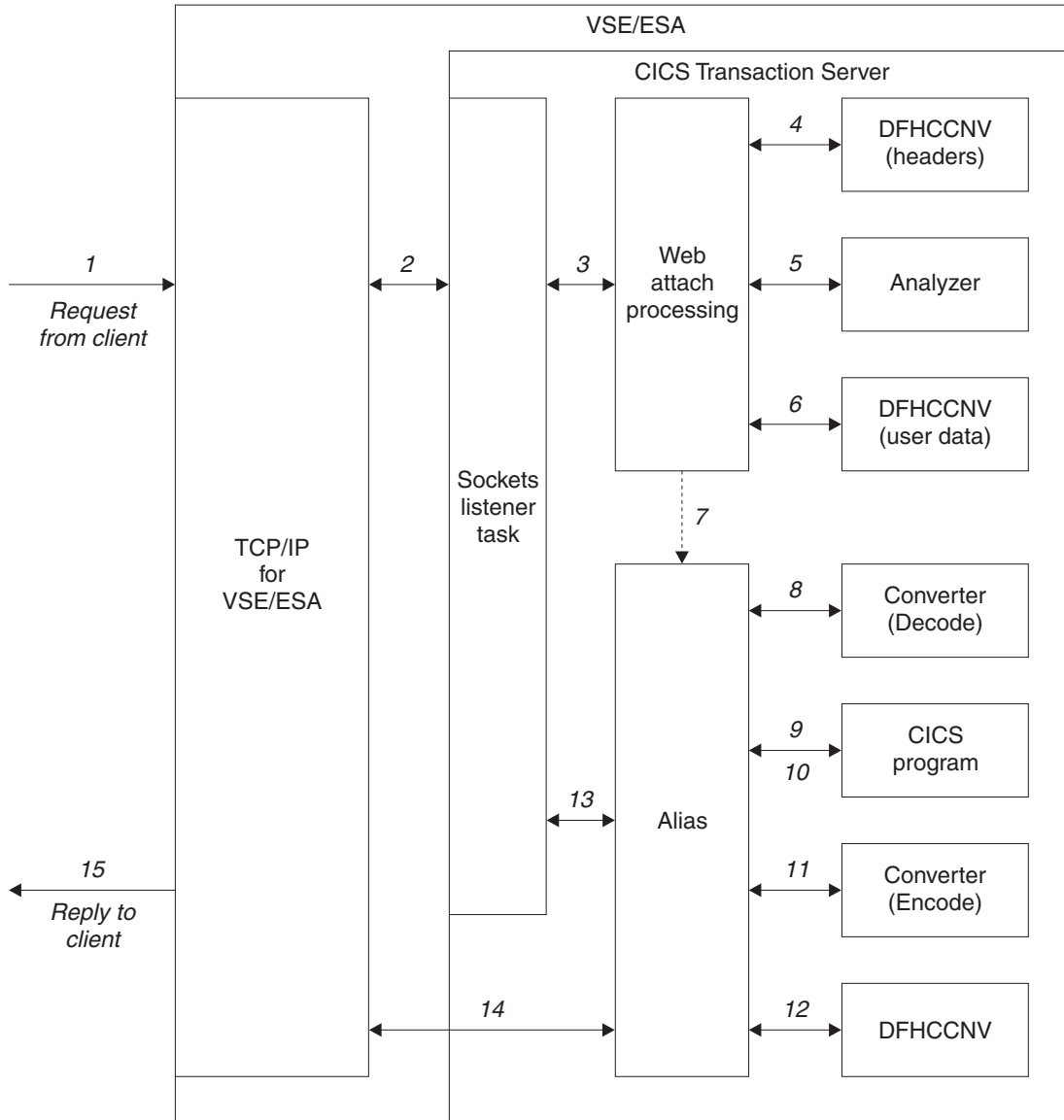


Figure 8. Calling a program with CICS Web support—control flow

1. The Sockets listener task monitors the TCP/IP for VSE/ESA interface for incoming HTTP requests, based on the port number specified on the TCPIP SERVICE resource definition.
2. An HTTP request arrives in TCP/IP for VSE/ESA from a Web browser.
3. The Sockets listener task attaches Web attach transaction CWXN. CWXN or its alias should be specified as the TRANSACTION on the TCPIP SERVICE definition.
4. Web attach processing receives the incoming request and calls DFHCCNV to translate HTTP request headers from ASCII to EBCDIC.
5. Web attach processing links to the user's analyzer (the analyzer URM).
6. If the analyzer requests conversion, Web attach processing calls DFHCCNV to translate the body of the HTTP request from ASCII to EBCDIC.
7. Web attach processing starts an alias transaction (CWBA or alias) to deal with all further processing of the request in CICS, then terminates.

8. If the analyzer requests a converter, the alias calls it, requesting the **Decode** function. **Decode** can modify the communication area for the CICS program. The converter is a user-written CICS program.
9. The alias calls the CICS program that the analyzer or **Decode** specified. The communication area passed to the CICS program is the one set up by **Decode**. If no converter program was called, the communication area contains the entire request.
10. The CICS program processes the request and builds a response using EXEC CICS WEB WRITE and EXEC CICS WEB SEND commands, or returns output in the communication area.
11. If the analyzer requested a converter, the alias calls the **Encode** function of the converter, which uses either the EXEC CICS WEB commands or the communication area to prepare the HTTP response. If no converter program was called, and no EXEC CICS WEB SEND command issued, the alias assumes that the CICS program has put the desired HTTP response in the communication area.
12. If the analyzer or application requested data conversion, the alias calls DFHCCNV to translate the HTTP response.
13. The alias returns the results to the Sockets domain, requests that the socket be closed, and returns.
14. The Sockets domain issues a call to TCP/IP for VSE/ESA to send the response.

Some variations on this process are possible:

- You might not use a CICS program, but construct the response in the **Decode** or **Encode** functions of the converter, or partly in both.
- You might not use a converter, but construct the response in the CICS program. In this case the CICS program must be written either to accept an HTTP request in its communication area, and to overwrite it with an HTTP response, or to use the Web-related CICS application programming interface to process an HTTP request and build an HTTP response.
- You might construct the response in the analyzer. In this case the alias does not call a converter, or a CICS program, but does the data conversion (if requested by the analyzer), and then sends the reply to the Web browser.

Dealing with non-HTTP requests

CICS Web support can be used to process requests that are not in the HTTP format. If the Web attach transaction cannot parse the incoming request as an HTTP request, the process illustrated in Figure 8 on page 19 is modified in various ways:

- There is no translation of any part of the request before it is passed to the analyzer. The analyzer must do its own translation, or work in the client code page.
- If the analyzer asks for data conversion, the whole of the data is translated before the alias is started.

Using CICS Web support to run a terminal-oriented transaction

Figure 9 on page 21 shows the control flow through CICS Web support for a request for a terminal-oriented 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 program to be called, followed by the name of the

transaction to be run. All BMS maps used must be recompiled and have HTML templates created.

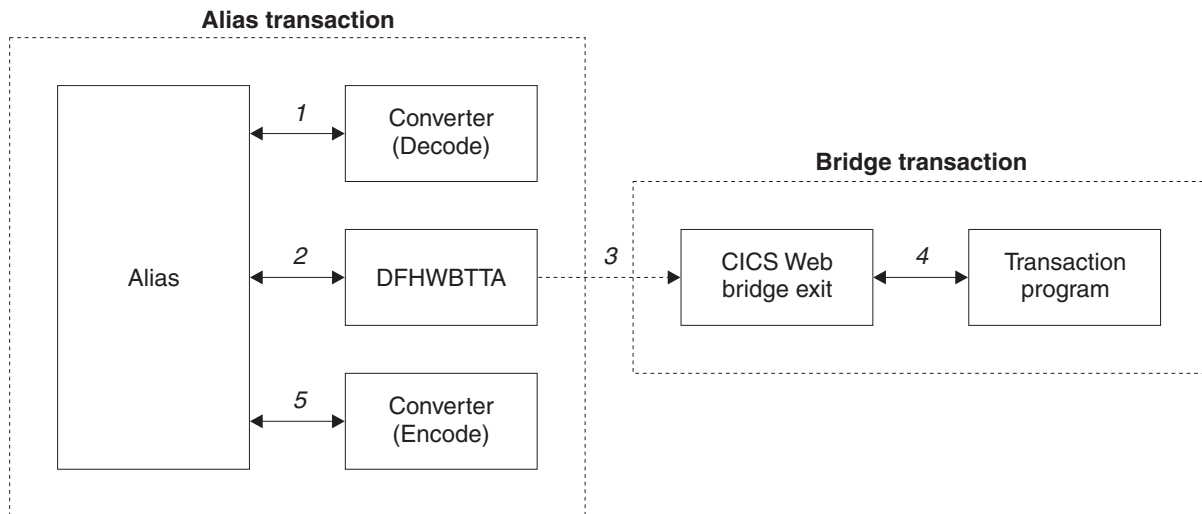


Figure 9. Running a transaction with CICS Web support—control flow

1. If the analyzer requests a converter, the alias calls it, requesting the **Decode** function. **Decode** sets up the communication area for DFHWBTTA. If no converter is specified, the 3270 Bridge will perform the BMS-to-HTML mapping.
2. The alias calls DFHWBTTA. The communication area passed to DFHWBTTA is the one set up by **Decode**. If no converter program was called, the communication area 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, DFHWBLT.
4. When the program attempts to write to its principal facility, the data is intercepted by the CICS Web bridge exit, and returned to the alias. If the caller requested a converter, the alias calls the **Encode** function of the converter, which uses the communication area to prepare the response. If no converter program was called, the alias assumes that the communication area contains the desired response.

Data flow in request processing

To make decisions about the facilities to use, and how to customize them, you need to understand how data is passed in the CICS Web support.

Using the CICS Web support commarea method to call a program

Figure 10 on page 22 shows the data flow from client through CICS and back to the client. As explained in “Using CICS Web support to call a program” on page 18, some of these steps are optional. See “Chapter 11. Writing CICS programs to process HTTP requests” on page 69 for more information about HTTP headers and HTTP requests.

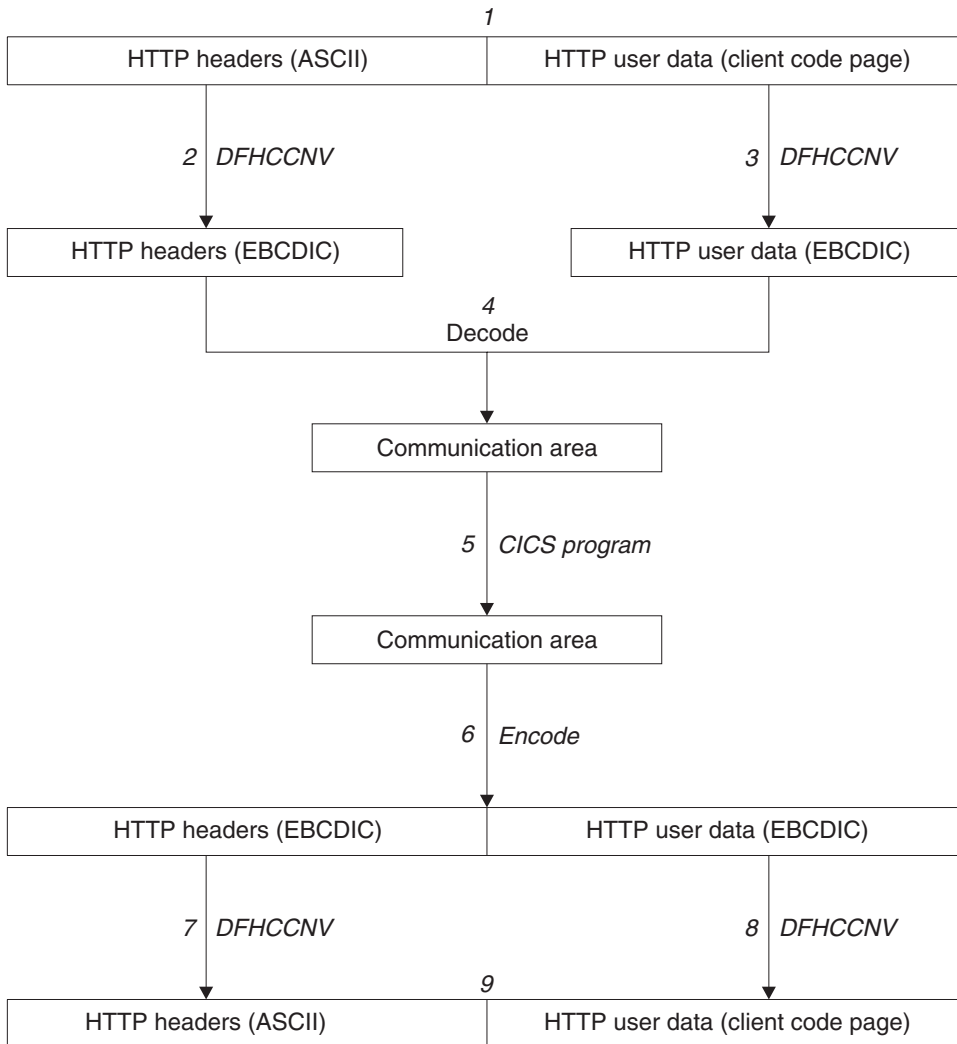


Figure 10. Calling a program using the CICS Web support commarea method—data flow

1. A request arrives from a client, and the CICS Sockets listener transaction, CSOL, starts the Web attach transaction, CWXN, and reads the request into CICS temporary storage.
2. DFHCCNV translates the HTTP headers from ASCII into EBCDIC.
3. DFHCCNV translates the HTTP user data from the client code page into EBCDIC.
4. The **Decode** function of the converter constructs the communication area for the CICS program. This communication area can be constructed in-place in the buffer provided by CICS. **Decode** can get a new buffer, or it can use the EXEC CICS WEB application programming interface to retrieve the parts of the incoming request.
5. The CICS program updates the communication area.
6. The **Encode** function of the converter constructs the HTTP response to be sent to the client. The response can be constructed in-place in the communication area. **Encode** can free the communication area and get a new buffer for the response, or it can use the new Web application programming interface to construct an HTTP response. The response consists of headers and user data. You can make your response longer than 32K, as described in “Explaining HTTP responses” on page 71.

7. DFHCCNV translates the headers from EBCDIC to ASCII.
8. DFHCCNV translates the user data from EBCDIC to the client code page.
9. The alias sends the response to the client, and frees the storage.

Chapter 4. Planning for CICS Web support

This chapter describes the planning tasks for CICS Web support. Major decisions about the kinds of requests you are going to allow and kinds of services you are going to provide are made here, and they affect the rest of the tasks involved in setting up CICS Web support.

Task	See...	Task completed?
Have you the correct prerequisites to use CICS Web support?	"Defining the prerequisites for using CICS Web support" on page 26	YES / NO
Which CICS transaction processing services are to be made available to users of CICS Web support and the CICS business logic interface? (These services can be CICS programs, or CICS transactions.)	For transactions, see "Chapter 9. 3270 applications on the Web" on page 49. For programs, see "Chapter 11. Writing CICS programs to process HTTP requests" on page 69.	YES / NO
How is your Web-related work going to be passed to CICS?	"Types of requester" on page 17.	YES / NO
(TCPIP SERVICE definitions form part of the processing of incoming requests.) Which CICS resources are to be accessed by which TCPIP SERVICE definitions.	See the <i>CICS Release Guide</i> for details of the TCPIP SERVICE definition, and "TCPIP SERVICE definitions" on page 31 for Web-specific considerations.	YES / NO
What level of security is required for each Web application?	"Chapter 13. Security for CICS Web support" on page 81	YES / NO
What is the URL format that you want to use for your applications to gain access to CICS services?	"URL format" on page 26	YES / NO
(A user-replaceable program known as the analyzer interprets incoming requests and is required for CICS Web support) Are you going to write your own analyzer or use the CICS supplied analyzer DFHWBADX?	"Chapter 6. Writing an analyzer for CICS Web support" on page 37	YES / NO
How to write a converter program to incorporate existing business logic into your Web application.	"Chapter 7. Writing a converter" on page 43	YES / NO
Are you going to use the communication area method or EXEC CICS WEB commands in your applications?	"Chapter 11. Writing CICS programs to process HTTP requests" on page 69	YES / NO
For CICS transactions that use BMS, decide what customization of the HTML output is necessary.	"Chapter 10. Creating HTML templates from BMS definitions" on page 57	YES / NO

Task	See...	Task completed?
What client and server codepages are going to be used?	“Code page considerations for Web API applications” on page 39, “Code page considerations for Web commarea applications” on page 39, “Defining a conversion table” on page 33, and “Appendix I. HTML-coded character sets” on page 149	YES / NO
If you use HTML templates or DOCTEMPLATES , where are they to be stored?	“DOCTEMPLATE definitions” on page 30	YES / NO
How long should CICS wait before deleting non-active resources ?	“Controlling web support with system initialization parameters” on page 29	YES / NO

Defining the prerequisites for using CICS Web support

This section describes the software requirements for using CICS Web support.

VSE/ESA

The following must be installed on the VSE/ESA system:

- TCP/IP for VSE/ESA. Ports belonging to TCP/IP for VSE/ESA must be made available for use by the CICS region involved.
- Language Environment. This provides the run-time libraries that are a prerequisite for running CICS Web support.

CICS

CICS must be set up for Language Environment support, as described in the *CICS System Definition Guide*.

TCP/IP for VSE/ESA

Ports belonging to TCP/IP for VSE/ESA must be made available for use by the CICS region involved.

URL format

- If requests are received by CICS Web support, the decision about URLs will affect the specification of the analyzer. “The default analyzer” on page 40 describes the conventions accepted by the default analyzer supplied with CICS Web support.
- If the requests are from other callers of the CICS business logic interface, you must decide for yourself what the caller must supply to request CICS transaction processing services. “Appendix A. Reference information for DFHWBBLI” on page 107 describes the communication area that callers must supply, and explains what the CICS business logic interface does with its input.

Operations tasks

- You can control the operation of CICS Web support by using CEMT or CPSM for the following resource types:
 - TCPIP
 - TCPIPSERVICE

and CEDA for these resource types:

- TCPIPSERVICE
- DOCTEMPLATE

See the *CICS Enhancements Guide* and the *CICS Supplied Transactions* for further information on these commands.

Chapter 5. Configuring CICS Web support

This chapter explains how to configure CICS Web support. Table 1 is a checklist of what you need to do.

Table 1. Configuring CICS Web support

Task	See...	Task completed?
Specifying the appropriate system initialization (SIT) parameters.	"Controlling web support with system initialization parameters"	YES / NO
Creating the necessary resource definitions.	"Defining resources to CICS"	YES / NO
Reserving ports for CICS Web support	"Reserving ports for CICS Web support" on page 35	YES / NO
Specifying a name server (optional).	"Specifying a name server" on page 35	YES / NO
Enabling lightpen support (optional).	"Enabling lightpen support" on page 36	YES / NO
Running the sample application to test CICS Web support.	"Running the sample application" on page 36	YES / NO

Controlling web support with system initialization parameters

CICS Web support is controlled initially by system initialization parameters. When CICS is running, you can make changes using CEMT and CEDA. There are four CICS system initialization parameters relating to CICS Web support:

- If you are using Web 3270 support, you can use the WEBDELAY parameter to fix:
 - 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 frequency, in minutes, with which the garbage collection transaction CWBG is run to delete the marked tasks and their data.
- The TCPIP parameter specifies whether CICS TCPIP services are to be activated at CICS startup. The default is NO, meaning that HTTP services cannot be enabled, and you cannot use any TCPIP SERVICE resources defined with CEDA.. If TCPIP is set to YES, HTTP services can be enabled and can then process work.

See the *CICS Enhancements Guide* for details of these new system initialization parameters, and the *CICS System Definition Guide* for general information on the SIT.

Defining resources to CICS

This section describes the resources needed to configure CICS Web support. It contains these topics:

- "CICS supplied resource definitions" on page 30
- "DOCTEMPLATE definitions" on page 30
- "TCPIP SERVICE definitions" on page 31
- "TRANSACTION definitions for extra alias transactions" on page 32
- "PROGRAM definitions for user-replaceable programs" on page 33

- “Setting up a sub-library for the template manager” on page 33
- “Defining a conversion table” on page 33

CICS supplied resource definitions

CICS Web support provides an RDO group defining the CICS resources used by the interface. The following definitions are in the locked group DFHWEB:

- Transactions required by CICS Web support (for example, CWBA and CWXN)
- Programs supplied with the CICS Web support

To change these definitions, you must copy them to your own RDO group and modify them there.

Sample CICS Web TCPIP SERVICE definitions are provided in the locked group DFH\$SOT. To change these definitions, you must copy them to your own group and change them there.

The group DFH\$WBSN contains the resource definitions for the security sample programs described in “Using sample programs for security” on page 82.

DOCTEMPLATE definitions

DOCTEMPLATE definitions allow you to perform variable substitution on documents in a manner similar to that done by BMS for 3270 screens. Templates can contain HTML, or binary data such as images. The data within the template is retrieved whenever a call is made for the template by means of an EXEC CICS DOCUMENT CREATE or EXEC CICS DOCUMENT INSERT command. The template can reside in any of the following places:

- VSE/ESA sub-library.
- CICS auxiliary temporary storage.
- CICS extrapartition transient data.
- CICS load module.
- CICS file.
- Exit program.

See the *CICS Enhancements Guide* for details of how to define a DOCTEMPLATE, and information about programming with documents and the associated EXEC CICS DOCUMENT commands.

VSE/ESA sub-library

You can use any editor you like to create the templates as members of this sub-library. The record format must be F (fixed). The templates can contain sequence numbers as follows:

- F format, and LRECL 80: the sequence numbers must be in record positions 73 through 80.

In any other case, there must be no sequence numbers in the records. The template manager decides whether there are sequence numbers by looking at the first logical record of a member of the VSE/ESA sub-library, so members that are only partially sequenced might be interpreted incorrectly. The default library name is DFHHTML. The sub-library name must be DFHDOC. The members must have a member type of HTML.

CICS temporary storage

Define one TSQUEUE for each template. The document handler domain returns an error if a request for a template is made to a non-existent TSQUEUE.

CICS transient data

Define an extrapartition TDQUEUE for each template. If you use an intrapartition transient data queue, your data is lost as soon as it has been read. If you use an extrapartition data queue, you must reset the queue after reading it.

CICS load module

Compile and link-edit a data-only load module. For example, an Assembler CSECT could contain a PROLOG containing your own control information, an ENTRY statement, any number of DC statements containing the HTML you want to output (you must put your own linefeeds in), and an END statement. CICS assumes that the entry point of the load module delimits the start of the template.

CICS file

This can be any CICS-controlled file.

Exit program

This is called whenever a request is made for the template. CICS passes a commarea to the exit program which is mapped by the following copybooks:

- DFHDHTXD (Assembler)
- DFHDHTXH (C)
- DFHDHTXL (PL/I)
- DFHDHTXO (COBOL)

The commarea contains the address (`dhtx_buffer_ptr`) and length (`dhtx_buffer_len`) of a CICS-supplied buffer in which the EXITPGM must return the template. The actual length of the template must be returned in `dhtx_template_len`. If the template to be returned is longer than `dhtx_buffer_len`, the template must be truncated to length `dhtx_buffer_len` and the EXITPGM must set the length required in `dhtx_template_len`. The EXITPGM is then called again with a larger buffer.

TCPIPSERVICE definitions

For HTTP requests to be submitted directly to CICS, you need one or more TCPIPSERVICE resources to be installed.

The TCPIPSERVICE definition allows you to define which TCP/IP services are to use CICS internal Sockets support. The internal CICS service that can be defined is CICS Web support.

The TCPIPSERVICE definition allows you to manage these internal CICS interfaces, with CICS listening on multiple ports, with different flavors of CICS Web support on different ports.

You must install and open a TCPIPSERVICE definition for each port on which CICS is to listen for incoming HTTP requests. You can create your own TCPIPSERVICE definition, or copy the HTTPNSSL definitions from the DFH\$SOT group into your own group and modify them to meet your system requirements.

The important parameters for a Web TCPIPSERVICE are:

- The STATUS must be OPEN
- The TRANSACTION to be attached by CICS when new work arrives on the specified port must be CWXN or a user-defined alias of CWXN, which must invoke DFHWBXN as the initial program.
- The port on which CICS is to listen
- The backlog of requests to be processed which TCP/IP for VSE/ESA is to allow

- The name of the analyzer user-replaceable module to be driven for TCPIPSERVICE
- An IP address on which CICS is to listen for incoming requests.
- A TS queue name. This is the 6-character prefix of TS queue names generated by CICS Web support when writing inbound and outbound data to temporary storage. If no prefix is supplied on the definitions, the default name of DFHWEB is used to generate TS queue names.

For more information on defining Web TCPIP SERVICES, see the *CICS Release Guide*.

TRANSACTION definitions for extra alias transactions

Two CICS transactions are provided with CICS Web support:

Web attach transaction (CWXN).

This CICS-supplied transaction invokes the analyzer program. It establishes the context in which the alias transaction CWBA is to run, and issues the appropriate ATTACH command. When CWXN is defined as the TRANSACTION on the TCPIP SERVICE definition, it is started by the sockets listener task CSOL when a new connection request is received on the port specified on the TCPIP SERVICE definition. If the HTTP 1.0 Keep-Alive header has been sent by the Web browser, CWXN remains in the system after the alias has been attached, and attaches new alias transactions to process further HTTP requests received from browser. If Keep-Alive has not been specified, CWXN terminates after the alias has been attached.

Alias transaction CWBA.

An alias transaction is a CICS-supplied transaction that is started by the Web attach transaction (CWXN) to process a single request. Many instances of the alias transaction can be active in a CICS system at the same time, each processing a different request. The alias transaction runs the CICS-supplied alias program that calls the CICS program. If you wish, you may set up additional transaction definitions for alias transactions, each using the CICS-supplied alias program.

You may want to use other alias transaction names for various reasons:

- Auditing
- Resource and command checking
- Allocating initiation priorities
- Allocating database plan selection
- Assigning different runaway values to different CICS programs

If you do want to use other alias transaction names, you must copy the definition of CWBA, making the necessary changes. The definition of CWBA is as follows:

```

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

```

You cannot change the program name in this definition. Only the CICS-supplied alias program DFHWBA can be used. All the extra alias transactions must be local transactions.

PROGRAM definitions for user-replaceable programs

Each incoming request is serviced by a CICS program that provides transaction processing services, and by two other user-replaceable programs, an analyzer (required) and a converter (optional).

If you are not using autoinstall for programs, you must define all the user-replaceable programs you use. If you are using autoinstall for programs, you do not need to define the converters. In any case analyzers must be defined with EXECKEY(CICS). All the user-replaceable programs must be local to the system in which CICS Web support is operating.

Setting up a sub-library for the template manager

If you use the HTML template manager for constructing HTTP responses, you may provide a VSE/ESA sub-library to hold the templates. You can use any editor you like to create the templates as members of this data set. The record format must be F (fixed). The templates can contain sequence numbers as follows:

- F format, and LRECL 80: the sequence numbers must be in record positions 73 through 80.

In any other case, there must be no sequence numbers in the records. The template manager decides whether there are sequence numbers by looking at the first logical record of a member of the VSE/ESA sub-library, so members that are only partially sequenced might be interpreted incorrectly.

Any VSE/ESA library can be used to specify sub-library member templates, as specified in the DOCTEMPLATE definition. If you are using the template manager (DFHWBTL) or the Web bridge (DFHWBTTA), references to templates that are not defined and installed as DOCTEMPLATE definitions are resolved as members of the library specified as DFHHTML. The sub-library name must be DFHDOC. The members must have a member type of HTML.

Defining a conversion table

If you have commarea-style Web applications which do not use the Web API, or you are using CICS Web support to run a terminal-oriented transaction, you need to create or modify a DFHCNV table for data conversion to allow CICS to deal with incoming requests. The use of the DFHCNV macro for defining the table is described in *CICS Family: Communicating from CICS on System/390*. There are two kinds of data conversion performed in CICS Web support:

Conversion of the HTTP header information. This information is always transmitted as ASCII data using the ISO 8859-1 (Latin-1) character set.

This is the base character set for HTTP and HTML. This data has to be translated into EBCDIC. The conversion template name that the server controller supplies to the DFHCCNV program, which does the translation, is DFHWBHH.

Conversion of the HTTP user data.

This information is transmitted in the code page of the HTTP client, and can be translated into EBCDIC if required. The conversion template name is supplied by the analyzer. If the request is not an HTTP request, all the request is translated using the name supplied by the analyzer.

For data conversion of the HTTP headers, you need to create a conversion template as follows:

```
DFHCNV TYPE=ENTRY, *
      RTYPE=PC, *
      CLINTCP=8859-1, *
      SRVERCP=037, *
      RNAME=DFHQBHH, *
      USREXIT=NO
DFHCNV TYPE=SELECT,OPTION=DEFAULT
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767, *
      LAST=YES
```

In the TYPE=ENTRY macro, the RNAME parameter must be DFHQBHH. The code page specifications CLINTCP and SRVERCP will get the HTTP request headers translated from ASCII to EBCDIC, and the HTTP response headers translated from EBCDIC to ASCII. The TYPE=SELECT and TYPE=FIELD macros must be coded exactly as shown.

For each name that the analyzer might specify for translating user data in the request from the client code page into EBCDIC, and for translating the user data in the response from EBCDIC to the client code page, you need to create a conversion template as follows:

```
DFHCNV TYPE=ENTRY, *
      RTYPE=PC, *
      CLINTCP=8859-1, *
      SRVERCP=037, *
      RNAME=DFHQBUD, *
      USREXIT=NO
DFHCNV TYPE=SELECT,OPTION=DEFAULT
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767, *
      LAST=YES
```

In the TYPE=ENTRY macro, the CLINTCP parameter must specify the code page of the client, and the RNAME parameter must specify the name that the analyzer will supply. The sample entry above supports translation of user data in the request from ASCII to EBCDIC, and of the user data in the response from EBCDIC to ASCII, for the default analyzer, which uses the name DFHQBUD. You may code the TYPE=SELECT and TYPE=FIELD macros in any way that is appropriate to the format of the user data that the client sends.

You may use the TYPE=INITIAL macro to set defaults for some of the values specified in these samples, as explained in *CICS Family: Communicating from CICS on System/390*.

The following sample shows a complete definition of the conversion templates for use with a Web browser using a Japanese double-byte character set. The code page 932 is one of several code pages for Japanese Web browsers, and 931 is one of the corresponding System/390® code pages. This sample can be used with the default analyzer. The sample must be assembled using the High-Level Assembler before being used.

```
DFHCNV TYPE=INITIAL
DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=DFHQBHH,USREXIT=NO, *
      SRVERCP=037,CLINTCP=8859-1
DFHCNV TYPE=SELECT,OPTION=DEFAULT
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767, *
      LAST=YES
DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=DFHQBUD,USREXIT=NO, *
      CLINTCP=932,SRVERCP=931
DFHCNV TYPE=SELECT,OPTION=DEFAULT
```

```
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,  *
      LAST=YES
DFHCNV TYPE=FINAL
END
```

Configuring TCP/IP for VSE/ESA

This section describes the changes you must make to TCP/IP for VSE/ESA as part of configuring CICS Web support.

Reserving ports for CICS Web support

You are recommended to reserve as many ports as you need for CICS Web support, and to ensure that CICS Web support has exclusive use of those ports.

Application programmers may use port numbers from 256 to 32 767 for nonstandard servers. For VSE/ESA, be careful when choosing port numbers less than 1024 — the “well-known” ports.

The maximum length of any queue of requests for a TCP/IP port on which a program is listening is controlled by TCP/IP for VSE/ESA, and is currently defined to be equivalent to a maximum of 10. This cannot be modified.

Identifying the TCP/IP server

You must identify the TCP/IP for VSE/ESA server by specifying a name for its IP address, otherwise CICS Web Support will not initialize successfully. You use the TCP command DEFINE NAME to do so. For example:

```
DEFINE
NAME,NAME=WINVSE.HURSLEY.IBM.COM,IPADDR=9.20.101.97
```

where NAME is a meaningful host name, and IPADDR specifies your server IP address.

If you do not do this, you will see the following error message during CICS start-up:

```
DFHS00117 applid Unable to determine the TCP/IP host name.
Language Environment return code X'00000458', return code X'00000000'.
TCP/IP services are unavailable.
```

Specifying a name server

If you want full CICS function (that is, if you want to use DFH\$WBSN and DFH\$WBENV), CICS Web support needs to access a name server during its operation. You set the name server in TCP/IP for VSE/ESA using the TCP command SET DNS1. For example:

```
SET DNS1=n.n.n.n
```

where *n.n.n.n* is the dotted decimal address of the name server.

To override the default search, you can code individual name, IP address pairs using the TCP command DEFINE NAME. For example:

```
DEFINE NAME,NAME=name,IPaddr=n.n.n.n
```

If the name server lookup fails when CICS runs:

- The security sample program DFH\$WBSN does not execute correctly.
- The environment variables program DFHWBENV does not return a connection name in SERVER_NAME, but the dotted decimal address of the connection, and it also returns a null string for REMOTE_HOST.

Enabling lightpen support

To enable selector pen processing over the CICS Web support 3270 bridge, you must define a bridge facility with lightpen support enabled. To do this, follow these steps:

1. Copy the following definitions to a new group. Unless all applications running on the CICS system require lightpen 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.

Running the sample application

A sample application DFH\$WB1A is provided to help you test the operation of CICS Web support. From a suitable Web browser, enter a URL that connects to CICS Web support with absolute path /CICS/CWBA/DFH\$WB1A. The response displays the message "DFH\$WB1A on system xxxxxxxx successfully invoked through the CICS Web support." with xxxxxxxx replaced by the application ID of the CICS system in which CICS Web support is running.

Chapter 6. Writing an analyzer for CICS Web support

The analyzer is a user-replaceable program for the CICS Web support. It interprets the incoming request and specifies the CICS resources that are needed to provide the requested service.

You must supply an analyzer, or use the IBM-supplied default analyzer DFHWBADX.

You can write your analyzer in Assembler, C, COBOL, or PL/I. Language-dependent header files, include files, and copy books are described in “Appendix B. Reference information for DFHWBADX” on page 115.

There is an analyzer for each CICS Web support TCPIP SERVICE. The place of the analyzer in CICS Web support is illustrated in Figure 8 on page 19. The analyzer can specify:

- The name of the CICS program that is to process the request
- The name of the converter that is to process the request
- The name of the alias transaction that is to process the request
- A user ID or terminal to be associated with the alias transaction
- Any code page conversion that is needed for user data
- A modified value for the user data length.

For reference information for the analyzer, see “Appendix B. Reference information for DFHWBADX” on page 115.

This chapter describes the analyzer program. It contains these sections:

- “Inputs”
- “Outputs” on page 38
- “Processing” on page 38
- “Code page considerations for Web API applications” on page 39
- “Code page considerations for Web commarea applications” on page 39
- “Performance considerations” on page 40
- “The default analyzer” on page 40

Inputs

The analyzer input includes:

- An eye-catcher for an analyzer parameter list
- The IP address of the client
- The IP address of the server
- An indicator of whether the request is an HTTP request

The analyzer input also includes the incoming request. If the request is an HTTP request, various parts of the request are identified by pointers and lengths to make processing easier.

These are the version, method, absolute path, and request header and have already been translated into EBCDIC by CICS. The user data is still in the client code page.

If the request is not an HTTP request (an FTP request for example), the input includes the entire request in the client code page. The pointers and lengths apply only to the communication area containing the first 32767 bytes of the incoming requests.

Outputs

The analyzer must provide the following output:

- A response code

It may also provide the following outputs:

- The name of the CICS program that is to service the request. If the request is for a terminal-oriented transaction, the program name must be DFHWTBTA.
- The conversion template name for code page translation of the user data.
- The transaction ID of the alias transaction that is to service the request.
- The name of the converter that is to be used to service the request.
- A user token that is to be passed to the converter functions.
- A modified value for the user data length.
- If the userid is not changed by the analyzer, the userid passed on input is used, if one was specified. If no userid is derived from anywhere, the CICS default userid is used.
- A reason code.

Processing

The inputs and outputs are presented in a CICS communication area. The analyzer can use any of its inputs to determine the CICS resources that are to be used to service the request, and the other outputs it might wish to supply.

To impose rules about which clients can use which services, you can use the input client IP address and the contents of the request to decide if this client is allowed to use this service. You can reject a client request by setting the output response to a value other than URP_OK.

You can specify a different analyzer for each TCPIP SERVICE, allowing the port number of the TCPIP SERVICE to determine which CICS resources are to process the request.

If the code page of the client is not an EBCDIC code page, you can set the output conversion template name. See “Code page considerations for Web commarea applications” on page 39.

If the request can be completely satisfied in the analyzer, you do not need to set the converter name or the CICS program name.

If the request can be completely satisfied by the analyzer and a converter, you must set the converter name, but not the CICS program name.

If you need a CICS program to service the request, and the program name can be determined by the analyzer, you should set the output CICS program name. (If you do not set it here, you must specify the use of a converter, and the converter **Decode** function must set the program name.)

If the selected CICS program needs a converter, you must set the output converter name.

If the service is to be provided under a user-defined alias transaction, you must set the output transaction name.

To pass any other information to the converter functions, you can set an output user token. This token could be a pointer to storage acquired with the SHARED option by the analyzer to be freed by the converter. You may also make changes to the contents of the request, and these are visible to **Decode** and to the CICS program. Any changes to the contents of the request held in the communication area are not reflected in the data returned by the EXEC CICS WEB commands.

If you want to use EDF to test your CICS programs, analyzers, or converters, you should use the CEDX transaction. The use of EDF is described in “Using EDF” on page 89.

You can use various return codes and reason codes to report errors in the inputs and processing. If the request is an HTTP request, some of the responses are associated with architected HTTP responses. For details consult “Appendix B. Reference information for DFHWBADX” on page 115. If you use any response other than URP_OK, or if you use any reason codes, you should document the responses and reason codes to help with problem determination.

If the request is a non-HTTP request, and you detect that there is more data to be received, you can use the URP_EXCEPTION response to request CICS to receive more data, and add it to that already in the input area. Web attach processing then calls the analyzer again.

Code page considerations for Web API applications

If you are using the EXEC CICS WEB and EXEC CICS DOCUMENT commands, you can specify the host and client codepages on the individual commands; these override any DFHCNV key allocated to this transaction by the analyzer.

For EXEC CICS WEB RECEIVE, the host codepage must be a server codepage supported by the CICS DFHCNV mechanism, and must therefore be set to one of the server codepage values listed in *CICS Family: Communicating from CICS on System/390*.

The client codepage must be one of those listed in “Appendix I. HTML-coded character sets” on page 149. You can specify either the IANA value or the IBM CCSID value, as CICS performs mapping between the two.

If there is an error during the processing of an HTTP request, and the Web error program is invoked, the DFHCNV key specified by the analyzer is used to determine what codepage conversion should be performed on the error response returned to the Web browser.

Code page considerations for Web commarea applications

When designing your analyzer, if you are not using the HTML base code page ISO 8859-1 (Latin-1) for user data, you need to specify the conversion template for the code pages used. You must perform the following steps:

1. Identify the character sets that HTTP clients will use. All the browsers that have access to the CICS Web support might use the same code page, or you might be able to tell the code page from the IP address of the client. It might be possible to get the browsers to create URLs that include an indicator of the

code page. The HTTP request headers Content-Type and Content-Language might contain useful information, but they are not used consistently by all web browsers.

2. Use *CICS Family: Communicating from CICS on System/390* to decide the kind of conversion to be performed, and add a conversion template to the DFHCNV table. For nonstandard conversion you need to create or modify the DFHUCNV program.
3. Write an analyzer that decides what data conversion is needed, and sets the name of the conversion template in the **wbra_dfhcnv_key** parameter.

If there is an error during the processing of an HTTP request, and the Web error program is invoked, the DFHCNV key specified by the analyzer is used to determine what codepage conversion should be performed on the error response returned to the Web browser.

Performance considerations

You should use performance-efficient techniques such as index tables to resolve the relations between request and CICS resources, rather than performing I/O operations. You should avoid allocating storage, since this can introduce processing delays.

CICS HTTP persistent connections support means that sockets connections with Web browsers can be kept open after the initial HTTP request has been processed. This has a significant effect on the amount of processing required for each HTTP request in the network. To enable CICS persistent connections support you must specify either NO or a numeric value for the SOCKETCLOSE keyword on the relevant TCPIP SERVICE definition. Note that CICS supports only the HTTP 1.0 Keep-Alive implementation of the persistent connections, not the HTTP 1.1 implementation.

To optimize the amount of processing required to retrieve a DOCTEMPLATE, you should consider storing the DOCTEMPLATES inside CICS or use a VSE/ESA sub-library on a virtual disk, rather than in a VSE/ESA sub-library on a real disk. The most efficient method of storing DOCTEMPLATES is as load modules, but the advantages of fast retrieval need to be weighed against the amount of CICS storage occupied by the template.

The default analyzer

DFHWBADX is the default analyzer for the CICS Web support. The source code for the analyzer is supplied in various languages, and you can use it as the basis of your own analyzer. The source files are as follows:

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

The default analyzer is written for HTTP requests in which the absolute path has one of the following five forms:

```
/converter/alias/program?token  
/converter/alias/program
```

/converter/alias/program/filename
/converter/alias/program/filename?token
/converter/alias/program/?token

The default analyzer links to the CICS-supplied utility DFHWBUN to unescape the user data in the communication area passed to the analyzer.

The default analyzer checks the eye-catcher, and then interprets the contents of the absolute path as follows:

- *converter* must be between one and eight characters long. It is converted to uppercase and interpreted as the name of the converter to be called by the alias, unless it has the value “CICS”, in which case the converter name is set to nulls to show that no converter is to be used.
- *alias* must be between one and four characters long. It is converted to uppercase and interpreted as the transaction ID of the alias transaction to be used to service the request.
- *program* must be between one and eight characters long. It is converted to uppercase and interpreted as the name of the CICS program that is to be used to service the request.
- *filename* can be any length, but it must not begin with a slash (“/”) or contain a question mark. It must be made up of characters allowed in URLs. It is ignored by the analyzer, but is available to the converter or the CICS program.
- *token*, a user-modifiable field. The first eight bytes are interpreted as the user token to be passed to the converter.

If *program* is DFHWBTTA, the *filename* is treated as the ID of a transaction to be run using the 3270 bridge facility. See “Chapter 9. 3270 applications on the Web” on page 49 for details of the interface to DFHWBTTA.

The default analyzer sets the conversion template name to DFHWBUD.

The default analyzer diagnoses various errors, and the meanings of its responses and reason codes are described in “DFHWBADX responses and reason codes” on page 119.

Chapter 7. Writing a converter

This chapter describes the converter.

You can have many converter programs in a CICS system to support the operation of CICS Web support. The place of converters in CICS Web support is illustrated in Figure 8 on page 19 and Figure 9 on page 21. The converter must run in the same CICS region as the TCPIP SERVICE which receives the request. Each converter must provide two functions:

- **Decode** is used before the CICS program is called. It can:
 - Use the data from the Web browser to build the communication area in the format expected by the CICS program.
 - Supply the lengths of the input and output data in the CICS program communication area.
 - Perform administrative tasks related to the request.
- **Encode** is used after the CICS program has been called. It can:
 - Use the data from the CICS program to build the HTTP response and HTTP response headers.
 - Perform administrative tasks related to the response.

It contains the following sections:

- “Writing a converter—Decode” on page 44
- “Writing a converter—Encode” on page 45

You might not need to write any converters. If the analyzer or 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 program in its communication area.

You may write your converters in Assembler, C, COBOL, or PL/I. Language-dependent header files, include files, and copy books are described in “Appendix C. Reference information for the converter” on page 121.

The converter provides **Decode** and **Encode** functions for processing a request.

There are some restrictions on what these functions can do when the converter is called from a CICS business logic interface that was called in offset mode. These are described below.

Performance considerations

The converter is called from the alias transaction, or from the CICS business logic interface, and therefore its functions can only affect the performance of a single client request.

You should avoid operations that introduce processing delays. If a converter function needs to allocate storage, it should use the NOSUSPEND option of EXEC CICS GETMAIN. The efficiency of later processing can be improved if **Decode** sets **decode_input_data_len** to the exact length of the data to be passed to the CICS program, since this optimizes the use of storage and data transmission facilities.

Writing a converter—Decode

This section gives informal descriptions of the inputs and outputs of **Decode**, and gives some hints about processing.

Inputs

The inputs to **Decode** include:

- An eye-catcher for a **Decode** parameter list
- The IP address of the client
- The name of the CICS program that is to service the request, if this was set by the analyzer, or the CICS business logic interface
- A pointer to the buffer containing the request (perhaps modified by the analyzer)
- The user token supplied by the analyzer, or by the caller of the CICS business logic interface
- A counter giving the number of times **Decode** has been entered in the current Web request. This is useful for loopback requests.

If the incoming request is an HTTP request, various parts of the request are identified by pointers and lengths to make processing easier:

- Version
- Method
- Absolute path
- Request header
- User data

Outputs

Decode must set the following outputs:

- A response code
- The length of the communication area to be passed to the CICS program

It might also provide the following outputs.

- A pointer to the communication area to be passed to the CICS program, if this is not the input communication area.
- The name of the CICS program that is to service the request.
- The user token to be passed to **Encode**.
- A reason code.

Processing

The main purpose of **Decode** is to provide the communication area for the CICS program.

- If your converter is running as part of the CICS Web support, or as part of the CICS business logic interface in *pointer* mode, the communication area passed to the target program can be the storage addressed by `DECODE_DATA_PTR` on entry to **Decode**, or you can use `EXEC CICS GETMAIN` to get new storage, and update `DECODE_DATA_PTR` to address the new storage. If `DECODE_DATA_PTR` is altered to address another storage location, it is the converter program's responsibility to freemain the original storage.
- If your converter is running as part of the CICS business logic interface in *offset* mode, the buffer must occupy the same storage as the input communication area. In this case you must not use `EXEC CICS GETMAIN` to get new storage, and you must not change the data pointer in the parameter list.

You can set the output for the length of the communication area you pass to the CICS program, and you can set an output for the returned length if this is less than the length to be passed to the CICS program.

You can use the input user token passed by the analyzer, and if this is a pointer, you can use and update the information in the storage it addresses. You can pass the same token on to **Encode**, or you can replace it with another token.

The CICS program name as set by the analyzer, or by the caller of the CICS business logic interface, is available for your use, and you can change it. If the program name has not been set already, you must set it here, or no CICS program will be called.

You can use various return codes and reason codes to report errors in the inputs and processing. If the request is an HTTP request, some of the responses and reason codes are associated with architected HTTP responses. For details consult “Appendix C. Reference information for the converter” on page 121. If you use any response other than URP_OK, or if you use any reason codes, you should document the responses and reason codes to help with problem determination.

However, if this occurrence of the decode converter is a loop back from the **Encode** converter, these pointers and lengths are set to zero (0), `DECODE_DATA_PTR` points to the request data from `ENCODE_DATA_PTR`, and `DECODE_INPUT_DATA_LEN` is the length in bytes of the data pointed to by `DECODE_DATA_PTR`. The user token is the same as it was from the exit of the **Encode** converter. On these secondary occurrences of the **Decode** converter, the user can still access the same information (using the `WEB EXTRACT` command) that the first occurrence could access. The user can detect whether this is a loopback request by checking the value of `DECODE_ENTRY_COUNT` and `ENCODE_ENTRY_COUNT`. Their value will be greater than 1 on a looped back request.

Writing a converter—Encode

This section gives informal descriptions of the inputs and outputs of **Encode**, and gives some hints about processing.

Inputs

The inputs to **Encode** include:

- An eye-catcher for an **Encode** parameter list
- A pointer to the communication area returned by the CICS program, and its length
- The user token created by the analyzer and passed by **Decode**
- A counter giving the number of times **Decode** has been entered in the current Web request. This is useful for loopback requests.

Outputs

Encode must set the following outputs:

- A response code
- A pointer to the buffer containing the response to be sent to the client

It might also provide the following outputs.

- A reason code

Processing

The main purpose of **Encode** is to provide the response to be sent to the client. You can use the HTML template manager to help you to construct the HTTP response; see “Appendix D. Reference information for DFHWBTL” on page 131. On exit from Encode, ENCODE_DATA_PTR must point to the buffer containing the response. You must set the output response length, and you must put the data length (response length plus 4) in the first word of the buffer.

- If your converter is running as part of the CICS Web support, or as part of the CICS business logic interface in *pointer* mode, the HTTP response can occupy the storage addressed by ENCODE_DATA_PTR on entry to Encode, or you can use EXEC CICS GETMAIN to get new storage and update ENCODE_DATA_PTR to point to the new storage. On exit from Encode, this new buffer must contain the HTTP response in the format described above.

If the request being processed was received by a CICS Web TCPIP SERVICE, and ENCODE_DATA_PTR has been altered to address another storage location, it is the converter program’s responsibility to freemain the original storage. CICS frees the storage addressed by ENCODE_DATA_PTR after the HTTP response has been sent. If the request being processed was not received by a CICS Web TCPIP SERVICE, 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 your converter is running as part of the CICS business logic interface in *offset* mode, the buffer must occupy the same storage as the communication area returned by the CICS program. In this case you must not use EXEC CICS GETMAIN to get new storage, and you must not change the data pointer in the parameter list.

You can use the input user token passed by **Decode**, and, if this is a pointer, you can use the information in the storage it addresses. If it is a pointer, you must use EXEC CICS FREEMAIN to free the storage it addresses.

You can use various return codes and reason codes to report errors in the inputs and processing. If the request is an HTTP request, some of the responses and reason codes are associated with architected HTTP responses. For details consult “Appendix C. Reference information for the converter” on page 121. If you use any response other than URP_OK, or if you use any reason codes, you should document the responses and reason codes to help with problem determination.

However, if the response code is URP_OK_LOOP, the CICS Web interface loops back to the **Decode** converter. The data pointed to by ENCODE_DATA_PTR should still be in the same format as a normal response (see “Appendix C. Reference information for the converter” on page 121 for reference information).

Chapter 8. The Web error program

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information. It describes the Web error program, DFHWBEP.

The Web error program — general

The Web error program, DFHWBEP, is a user-replaceable module driven by CICS Web support when there is a failure in the processing of a Web request received by a CICS Web TCPIP SERVICE. DFHWBEP allows you to modify the HTTP response issued by CICS, or to put out an alternative message.

The parameter list passed to the Web error program contains a pointer to a buffer containing the default HTTP response returned by CICS for the error detected, and the length of the response. The Web error program can:

- leave the response unchanged.
- modify the response to be returned, and update the length in `WBEP_RESPONSE_LEN` accordingly.
- GETMAIN a new buffer, build a new HTTP response, and pass back the address of the new buffer in `WBEP_RESPONSE_BUFFER` and the length of the new response in `WBEP_RESPONSE_LEN`.

The EXEC CICS WEB application programming interface is not available from the Web error program. The data to be returned to the client must be in the buffer addressed by `WBEP_RESPONSE_PTR`.

The default HTTP response is passed to the Web error program in its EBCDIC form. CICS assumes that the HTTP response addressed by `WBEP_RESPONSE_PTR` on exit from the Web error program is in EBCDIC, and performs codepage conversion on the response to convert it to ASCII before returning it to the client. The key used for this conversion is that selected by the analyzer user-replaceable module. If none was selected, or the analyzer was not invoked before the error occurred, the response is assumed to be in the ISO-8859-1 codepage.

If the error being processed is a sockets send or receive error, no error response is returned to the browser before closing the socket.

Inputs

Input to DFHWBEP is:

- Name of target program
- Name of program in which error occurred
- Abend code
- Associated message number
- Pointer to HTTP response to be returned. DFHWBEP can overwrite the CICS Web support response with its own HTTP response, which might be more meaningful to users.
- Length of HTTP response. The maximum length of the response is 32K.
- Server and client IP address
- Error code identifying the nature of the error
- Response and reason codes returned by the analyzer or the converter program.

Outputs

DFHWBEP returns a user-defined HTTP response and its accompanying text to the client.

Processing

The main purpose of the Web error program is to allow the CICS system administrator to customize or tailor the default HTTP error response returned by CICS for the error detected.

This ensures that the response that appears on the Web browser is meaningful to the user.

For reference information for DFHWBEP, see “Appendix H. Reference information for DFHWBEP” on page 145. For more information on user-replaceable modules, see the *CICS Customization Guide*.

Chapter 9. 3270 applications on the Web

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.

DFHWBTTA is a callable CICS-supplied program that provides an interface between Web browsers and CICS transactions. DFHWBTTA and its associated programs perform the translation between HTML and 3270 data streams or BMS maps. DFHWBTTA supports non-conversational, conversational, and pseudoconversational transactions.

This chapter is intended for programmers who write converters that create or modify requests to run CICS transactions, and for callers of DFHWBTTA who use the CICS business logic interface.

Input to DFHWBTTA

The communication area for DFHWBTTA must contain an HTTP request for a CICS transaction. There are two types of requests:

1. Initial requests are requests that are not continuations of conversations or of pseudoconversations. The request for the first transaction of a sequence of transactions in a pseudoconversation is an initial request. The first request for a conversational transaction is an initial request. The only request for a transaction that is neither conversational nor pseudoconversational is an initial request.

To send data on the initial request, use plus signs (+) rather than blanks to separate the transaction id and any further data. For example, to start transaction CEMT with the string CEMT INQ TAS, use the following path on the URL:

```
/cics/cwba/dfhwbttta/CEMT+INQ+TAS
```

CICS passes this data to the 3270 application in the form of a formatted 3270 datastream. The initial path can be in any format, as long as the transid follows the last "/". The form used on subsequent HTTP responses for the same Web 3270 conversation uses the same path that was input to DFHWBTTA.

2. Continuation requests are requests that continue a conversation or pseudoconversation. DFHWBTTA retains information about conversations and pseudoconversations that allows it to recognize a request as being a continuation request. Identification of the retained information is passed in a hidden variable in the HTML generated for the previous request.

The request must be encoded in EBCDIC. The format of the URL in the HTTP request must be in one of the following four forms:

```
/converter/alias/program/tranid
```

```
/converter/alias/program/tranid?token
```

```
/converter/alias/program/keyword/tranid
```

```
/converter/alias/program/keyword/tranid?token
```

- *converter* must be between one and eight characters long. It is ignored by DFHWBTTA.

- *alias* must be between one and four characters long. It is ignored by DFHWBTTA.
- *program* must be between one and eight characters long. It is ignored by DFHWBTTA.
- *keyword* is optional and is not case sensitive. If it is present, it must have the value UNFORMAT. If the UNFORMAT option is present, DFHWBTTA assumes that the transaction id has been entered from an unformatted screen.
- *tranid* can be of any length. For an initial request, DFHWBTTA interprets it as a transaction ID. For a continuation request, it is ignored.
- *token*, if it is present, is ignored by DFHWBTTA.

A continuation request may also contain user data. This user data must consist of URL-encoded data. URL-encoded data is data in the form of *variable=value* elements separated by ampersands. The data is to be interpreted as a BMS map, or as a 3270 data stream. The map or data stream is expected to be the browser's response to the previously output HTML. The variables are interpreted as follows:

- Retained data for a continuation request. The value of the variable DFH_STATE_TOKEN identifies the retained data for the continuation request.
- Cursor position. The value of the variable DFH_CURSOR is interpreted as the name of the field in which the cursor is to appear. The corresponding cursor position is passed to the application program in EIBCPOSN.
- AID indicator. The first occurrence of any of the following variables defines the AID that will be passed to the application: DFH_ENTER, DFH_CLEAR, DFH_PF1, ..., DFH_PF9, DFH_PF10, ..., DFH_PF24, DFH_PA1, ..., DFH_PA3, DFH_PEN. The values associated with these variables are not significant in the conversion of the data to a BMS map or 3270 data stream.
- Data fields. Each of the fields of the BMS map is represented by a variable whose value is interpreted by DFHWBTTA as the value of the data supplied by the browser. The name of each variable is the same as the name of the field in the BMS map.
- Modified field indicators. Variables of the form DFH_NEXTTRANSID.*n*, where *n* is a number, specify the names of the modifiable fields that will be searched to find a transaction ID. The values of these variables are the names of other variables in the URL-encoded data.

For a continuation request, DFHWBTTA determines the transaction ID as follows:

- If the request is part of a pseudoconversation, and the previous transaction ended with RETURN IMMEDIATE TRANSID=, the specified transaction ID is the one that is used.
- If the request is part of a pseudoconversation, and the previous transaction ended with RETURN TRANSID=, the specified transaction ID is the one that is used.
- If the request is part of a pseudoconversation, but the previous transaction did not specify a transaction ID on the RETURN command, but the AID is associated with a transaction ID, that transaction ID is used.
- If the request is part of a pseudoconversation, but no transaction ID was specified on the RETURN command, and there is no transaction ID associated with the AID, then the first four bytes of data in the first modified field are taken to be the transaction ID. If the data in the modified field has a blank in the first four bytes, the transaction ID is the data up to the first blank. The method of determining the first modified field is as follows:
 1. Set *n* to 1.
 2. Search for DFH_NEXTTRANSID.*n*.

3. If there is no occurrence of DFH_NEXTTRANSID.*n*, end the search.
 4. If there is an occurrence of DFH_NEXTTRANSID.*n*, search for the variable whose name is the value of DFH_NEXTTRANSID.*n*.
 5. If there is such a variable, use the value to determine the transaction ID.
 6. If there is no such variable, add 1 to *n*, and return to step 2.
- If the request is part of a conversation, then the waiting transaction is continued.

Customizing the input to DFHWBTTA

The HTTP request is prepared by the **Decode** function of the converter, if the caller asks for a converter. The converter may make modifications to the request.

On input to **Decode**, exactly one of the AID variables is present in the user data, and it is the one set by the browser. You can insert your own AID variable, or modify the existing AID variable.

You can modify information about the cursor position by changing the value of DFH_CURSOR. The value of DFH_CURSOR must be the name of one of the variables that define the contents of the data fields. The standard technique for generating HTML pages from BMS maps produces HTML pages that track cursor movements in the Web browser, and report the final position of the cursor in DFH_CURSOR.

You can insert or delete DFH_NEXTTRANSID.*n* variables to control the selection of the next transaction ID that is described in “Input to DFHWBTTA” on page 49. If you add an instance of DFH_NEXTTRANSID.*n*, use the name of one of the other variables as the value of DFH_NEXTTRANSID.*n*.

Decode must not modify the value of DFH_STATE_TOKEN.

Output from DFHWBTTA

DFHWBTTA presents an HTTP response to the **Encode** function of the converter (if any). 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. This output might have been customized as described in “Chapter 10. Creating HTML templates from BMS definitions” on page 57.

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 user data>
- 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.

Customizing the output from DFHWBTTA

If you are using the functions of DFHWBTTA that emulate the non-BMS terminal commands, you can modify the appearance of the generated page by providing header and footer information for the page. The main part of the 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. This screen image is not directly customizable, unless you choose to modify it in your ENCODE converter function (see “Customizing with Encode” on page 54). However, you can specify HTML to be inserted before and after this screen image representation by installing document templates containing customized markup. You supply one or more of the following templates, whose names are defined in the TEMPLATENAME fields of DOCTEMPLATE definitions:

*tran*HEAD

This is a template that is inserted at the head of the HTML page being output for transaction *tran*, if it is installed.

CICSHEAD

This is a template that is inserted at the head of the HTML page being output for transactions that do not have a corresponding *tran*HEAD template installed.

*tran*FOOT

This is a template that is inserted at the foot of the HTML page being output for transaction *tran*, if it is installed. If this template is not installed, CICSFOOT is used instead.

CICSFOOT

This is a template that is inserted at the foot of the HTML page being output for transactions that do not have a corresponding *tran*FOOT template installed.

The HTML generated to represent the screen image is designed to be presented in a non-proportional font, so that the column alignment implied by the 3270 screen addresses is approximately preserved. CICS generates a `<pre>` tag at the beginning of the page for you, but you should generate the closing `</pre>` tag yourself in your customized footing template (*tran*FOOT or CICSFOOT). These tags ensure that the screen image is successfully generated in a non-proportional font.

Required contents for a heading template

If you choose to supply heading or footing templates, you must supply some of the required elements of an HTML page. 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 *<title>* tag. For example:

```
<title>A sample title</title>
```
- 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">
```


- Optionally, any masthead images, heading tags, navigational links, or anything else needed to create your customized page.

The default header generated by CICS is as follows:

```
<!doctype html public "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>CICS Web support screen emulation</title>
<script language="JavaScript">
</script>
<meta name="generator" content="CICS Transaction Server/1.1.1">
</head>
<body>
```

Required contents for a footing template

If you choose to supply heading or footing templates, you must supply some of the required elements of an HTML page. A footing template should contain the following HTML elements:

- A `</pre>` tag, to terminate the non-proportional text begun by CICS. If you do not specify a `</pre>` tag, any input buttons you specify are displayed vertically rather than horizontally.
- 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 “Input to DFHWBTTA” on page 49, 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 customizations of your pages
- A `</body>` tag to close the page
- An `</html>` tag

If you do not specify a footing template, the CICS-generated footing contains buttons for all the possible AID indicators; this may not be suitable for your customized page.

The default footer generated by CICS is as follows:

```
</pre>
<input type="submit" name="DFH_PF1" value="PF01">
<input type="submit" name="DFH_PF2" value="PF02">
<input type="submit" name="DFH_PF3" value="PF03">
<input type="submit" name="DFH_PF4" value="PF04">
<input type="submit" name="DFH_PF5" value="PF05">
<input type="submit" name="DFH_PF6" value="PF06">
<input type="submit" name="DFH_PF7" value="PF07">
<input type="submit" name="DFH_PF8" value="PF08">
<input type="submit" name="DFH_PF9" value="PF09">
<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>

```

Customizing with Encode

The **Encode** function may make changes to the response. If the transaction is expecting a response from the user (either conversational or pseudoconversational), the changes to the output must still allow the continuation request to be correctly understood by the next part of the conversation or pseudoconversation.

Lightpen operation

CICS Web support allows applications that support selector pens to be run from a Web browser. (For details of selector pens, see the *CICS Application Programming Guide*.) To do this, the bridge facility associated with the transaction must be properly configured, as explained in “Enabling lightpen support” on page 36. CICS Web support recognises a field as being detectable only if:

- the field attribute byte identifies the field as being detectable or intensified (that is, bright), and
- the first character of the field contains a valid designator character. This can be an ampersand (&), a right angle bracket (>), a question mark (?), a blank, or a null.

When a field is determined to be selector pen detectable, the field appears on the browser with a checkbox preceding it. The designator character, which would have appeared as the first character in the field on a 3270 device, is removed from the field data, and only the remaining characters are displayed. The field length on the browser is decreased by one character.

The checkbox contains a check symbol (✓) only if the designator character is a right angle bracket (>); to select a field, check or uncheck the checkbox accordingly.

If the field is a selection field, checking and unchecking the checkbox simulates toggling the modified data tag (MDT) bit on and off. If you uncheck the checkbox on an unprotected field and enter data, the MDT bit is not switched on. (Note that this is different to what happens on a 3270 device.)

If the field is an attention field, checking the checkbox does not cause data to be transmitted to the CICS region. To do this, check the checkbox associated with the attention field and click on the button marked “Pen”. If multiple attention fields are

checked, the attention field closest to the screen origin (that is, row 1 column 1) will be used as the attention field. if no attention field is checked, CICS assumes that the ENTER key has been pressed.

Chapter 10. Creating HTML templates from BMS definitions

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.

This chapter describes how to create HTML templates from an existing BMS mapset definition. You can generate templates using standard generation, or customized generation.

Source for BMS mapsets that are available only as load modules can, with some limitations, be recovered using DFHBMSUP. See *CICS External Interfaces Guide* for details.

Generating a standard template

A template generated by the standard method contains the following:

- Constants and input fields from the map
- Buttons to represent the following:
 - ENTER key
 - PA1, PA2, and PA3 keys
 - Program function keys PF01 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 9. 3270 applications on the Web” on page 49.
- A hidden variable DFH_CURSOR whose value is the name of the field in which the cursor is set in the map.
- A JavaScript function dfhsetcursor, which when invoked in the browser sets the cursor position to the field whose name is the value of DFH_CURSOR.
- A JavaScript exception handler for the onLoad exception. This function invokes dfhsetcursor, and tracks the movement of the cursor.

Why customize the generation of templates?

There are many reasons why you might wish to change the output from generating a template for a BMS map. You can:

1. Support the application's use of keys that are not in the standard output.
2. Suppress the HTML Reset function, which does not correspond to any 3270 function.
3. Change the appearance of the keys, or the text associated with them.
4. Provide an HTML title for the HTML page.
5. Provide a masthead graphic for the HTML page.
6. Change the color of the background, or specify a special background.
7. Modify the BMS colors. You might need to do this if the BMS colors do not show up well against the background.
8. Suppress parts of the BMS map.
9. Add Web browser control functions, JavaScript functions for example, to the HTML page.
10. Add text that appears only on the HTML page, but is not part of the BMS map.
11. Add HTML header information to the HTML page.

Examples of these customizations are given in “Customization examples” on page 59 .

Customization facilities

There are two facilities provided to help you customize the HTML templates:

1. The DFHMDX macro (invoked from within DFHMSX): You use the DFHMDX macro to define your own customization macro that is used when the templates are being created from the BMS map definitions. You use a customization macro for the customizations numbered 1 to 9 in the list in “Why customize the generation of templates?” on page 57.
2. The DFHWBOUT macro: You add invocations of the DFHWBOUT macro to the BMS map definitions. This macro inserts text in the HTML page, and you use it for the customizations numbered 9 to 11 in the list in “Why customize the generation of templates?” on page 57.

(For customization number 9 (above) you have to coordinate what you put in the customization macro with what you put in DFHWBOUT.)

How to produce the HTML templates

The procedure is as follows:

1. Review the 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 LIBDEF source 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 mapset being processed. The HTML template names consist of the label from the DFHMSD macro plus one character starting from ‘A’. 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 mapset 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 LIBR source statements that set up one template for each map in a mapset.
4. Use LIBR to store the templates in the template library. You must store the templates in a VSE/ESA sub-library named DFHDOC, although the library name can be any valid library name.
5. If you want to put your templates in a VSE/ESA sub-library other than DFHHTML.DFHDOC, you must define DOCTEMPLATE definitions for your templates, and specify an alternate library.

Restricting the size of HTML templates

If the template is to be used by a transaction run using the 3270 Bridge, the size of the template is restricted. If the template requires more than 32K of storage to be read from the VSE/ESA sub-library, any attempt to use the 3270 Bridge results in message DFHWB0133 being issued with a code of X'4119'.

Even if the template requires less than 32K of storage it can still cause an error 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 VSE/ESA sub-library. It also issues warning messages when the size of the template exceeds 30K and 32K.

Writing a customizing macro definition

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. 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 63, and its use is described in “Customization examples”. DFHMDX is invoked from within DFHMSX.

4. A MEND statement to end the definition.

Customization examples

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 mapset name and map name. Later invocations override or add to the parameters for specific maps in the mapset. The continuation characters are in column 72, and the continued text is resumed in column 16.

```
MACRO
DFHMSX
DFHMDX MAPSET=*,MAP=*,
      PF1='Help,PF3='Exit',PF4='Save',PF09='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 DFHMSX customizing macro. Each DFHMDX macro is processed in sequence, and if applicable, the parameter values are stored. Where a duplicate parameter is specified for a particular map or mapset, the new value replaces the previous value for that map or mapset only.

The first DFHMDX macro in this example, where MAPSET=* and MAP=*, specifies a value of "Exit" for the PF3 keyword of any subsequent occurrence of DFHMDX. This value applies to every mapset and map in every subsequent DFHMDX macro until a new value is specified for the PF3 keyword. Here, PF3 remains as the 'Exit' key for all mapsets and maps until it is set to "Messages" for all the maps in mapset DFHWB0. It is then set to "Save" for map DFHWB02 only; in all the other maps in DFHWB0, PF3 is still "Messages", and in all mapsets and maps other than DFHWB0, PF3 is still "Exit".

The customizations listed in “Why customize the generation of templates?” on page 57 can be performed as follows:

1. **Support the application’s use of keys that are not in the standard output.**

You can add a key to the map AD001 as follows:

```
DFHMDX MAP=AD001,PF18='Resubmit'
```

The Web browser displays a key with the legend “Resubmit”. If the user presses this key, it is reported to the application as PF18.

2. **Suppress the HTML Reset function.**

You can suppress the Reset function for the map AD001 as follows:

```
DFHMDX MAP=AD001,RESET=NO
```

The Web browser displays a page that does not contain a Reset key.

3. **Change the appearance of the keys, or the text associated with them.**

You can change the legend on the PF1 key as follows:

```
DFHMDX PF1='Help'
```

The Web browser displays a key with the legend “Help”. If the user presses this key, it is presented to the application as PF01.

4. **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 browser displays “CICS Web Interface” as the title of the page.

5. **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 browser uses the specified masthead, or will show “CWI” as the masthead if it cannot find the graphic file.

6. **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 browser uses the specified file as a background for the page.

To change the color of the background, use the BGCOLOR parameter.

7. **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 browser shows BMS blue text in HTML aqua (the same as BMS turquoise), and BMS yellow text in bright orange.

8. **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.

9. Add Web browser 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:

```
DFHMDX MAP=AD001,ONLOAD='jset(''CWI is wonderful'',''Hello there!'')
```

will get the JavaScript function jset 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!".

10. 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 in the place that you want the text to appear in. 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.
```

11. 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="&WBDAT&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 &WBDAT and &WBTIME to the time and date at which the macro is assembled.

12. Using 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 codepages. For example:

```
DFHMDX OPENSQ=[,CLOSESQ=],OPENBR={,CLOSEBR=},EXCLAM=!
```

This specifies the substitutions needed. The characters must be entered on a terminal where the codepage corresponds to the SERVERCP on the DFHCNV call.

HTML and browser considerations

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 datastream is converted into HTML, the browser interpretation of this generates different output to that found on a 3270 terminal.

A string of blanks is ignored by a browser 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 browsers handle them inconsistently. They may or may not be displayed.

Limitations

CICS Web 3270 supports the following terminal control commands:

- EXEC CICS SEND (but not the STRFIELD option)
- EXEC CICS CONVERSE (but not the STRFIELD option)
- EXEC CICS RECEIVE

It also supports minimum function BMS and the EXEC CICS SEND TEXT command.

The following limitations apply to CICS Web 3270 support:

- The ATTRB=BRT option of a BMS field has no effect for an unprotected (input) field. This applies if the field is defined with ATTRB=BRT in the map definition or if the field attribute is changed to BRT dynamically on an EXEC CICS SEND MAP command.
- If a BMS program changes the attribute of a field in the map dynamically (by moving a 3270 attribute value to the attribute byte of a field named in the logical map), this change is not reflected in the HTML template subsequently sent to a browser. The template is sent as it is defined in the template dataset.
- The emulation of lightpens is not supported.
- There is no support for partitions, logical devices codes, magnetic slot readers, outboard formatting, or other hardware features.
- EXEC CICS DEFRESP is ignored. This may affect application recovery.
- The COLOR option is not supported for terminal control commands.
- User transactions can mix BMS and non-BMS requests, subject to the following restrictions. Transactions not following these guidelines will abend AWC3:
 - A BMS RECEIVE must follow a BMS SEND.
 - A terminal control RECEIVE must follow a terminal control SEND.
 - To change from using BMS requests to using non-BMS requests and vice versa, use a SEND with the ERASE option.

These restrictions apply from one transaction to the next in a pseudo-conversation. This means that if a transaction issues a SEND MAP and

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 mapset.

MAP specifies the name of the map within the mapset 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 browser 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.

GREEN

specifies the color to appear in the HTML page where green is specified in the BMS map. The default is #008000.

NEUTRAL

specifies the color to appear in the HTML page where neutral is specified in the BMS map. The default is #000000.

PINK specifies the color to appear in the HTML page where pink is specified in the BMS map. The default is #FF00FF.

RED specifies the color to appear in the HTML page where red is specified in the BMS map. The default is #FF0000.

TURQUOISE

specifies the color to appear in the HTML page where turquoise is specified in the BMS map. The default is #00FFFF.

YELLOW

specifies the color to appear in the HTML page where yellow is specified in the BMS map. The default is #FFFF00.

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.

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 information in the template. Use the keyword FOOT to suppress the footer information in the template. If you want to create a single HTML page from several maps, specify FOOT on the first, HEAD on the last, and both FOOT and HEAD on all the maps in between.

If you wish 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' (codepage 37).

CLOSESQ

The hex value or the character to be used to display a close square bracket. The default is x'BB' (codepage 37).

OPENBR

The hex value or the character to be used to display an open brace. The default is x'CO' (codepage 37).

CLOSEBR

The hex value or the character to be used to display a close brace. The default is x'D0' (codepage 37).

EXCLAM

The hex value or the character to be used to display an exclamation mark. The default is x'5A' (codepage 37).

CODEPAGE

specifies the IBM codepage number in which any text generated by the template generation process is encoded. This codepage must match the codepage 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 IBM host codepages supported by CICS are described in *CICS Family: Communicating from CICS on System/390*. The default codepage is 037.

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 PF01 to PF24, PA1 to PA3, CLEAR, ENTER, and PEN.

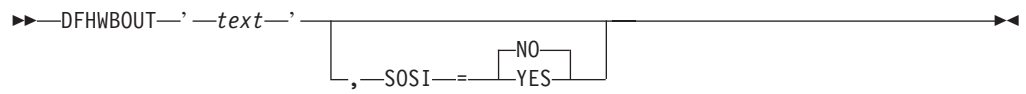
button can be (IMAGE, *url*), where *url* specifies the URL of a graphic image to be used for the button, or '*text*', where *text* is the text to be put in the button, or NO if the button is not to appear.

bmscolor can be any of BLUE, GREEN, NEUTRAL, PINK, RED, TURQUOISE, and YELLOW.

The DFHWBOUT macro

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 inline in the HTML page immediately following the text generated by the preceding DFHMDF macro.

DFHWBOUT



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.

Chapter 11. Writing CICS programs to process HTTP requests

This chapter describes facilities that help you to write CICS programs that process HTTP/1.0 requests and responses. Note that unpredictable results may occur if you use HTTP/1.1-specific headers.

- “Using HTTP requests” describes HTTP requests and how CICS handles them.
- “Explaining HTTP responses” on page 71 describes HTTP responses and how a CICS program can construct them.
- “Sample application programs” on page 76 describes a small sample program that you can use to test the operation of CICS Web support.

A CICS Web program can communicate with its caller by means of a CICS communication area. If the program is supported by a converter, the communication area contains the information put in it by **Decode**, otherwise it contains the entire HTTP request. The HTTP header information is in EBCDIC, and if the analyzer asks for data conversion, the user data has been translated using the analyzer-specified key.

Using HTTP requests

This section gives an outline of the formats of HTTP requests. Detailed information can be found in the references in “Information on the World Wide Web” on page 165.

A Web resource is identified by a uniform resource locator (URL), which identifies the host, and the resource requested. A user of a Web browser can enter a URL like the following:

```
http://www.ibm.com:80/Scripts/Global/nph-cc?cc=at
```

In this URL,

- `www.ibm.com` is the name of the host to which the request is to be sent.
- `80` is the TCP/IP port to which the request is to be sent. (80 is the default port for HTTP, and is not usually specified.) If the port is omitted, so is the colon that precedes it.
- `/Scripts/Global/nph-cc` is the absolute path, identifying a file to be retrieved, or a CGI script to be executed.
- `cc=at` is the query string.

The URL is converted by the browser into an HTTP request. An HTTP request consists of a request line followed by zero or more HTTP headers, each delimited by a carriage return line feed (CRLF), followed by optional user data. An HTTP header consists of a name, a colon, a space, and a value. An additional CRLF delimits the headers from the user data. The HTTP request line derived from the sample URL above contains:

```
GET /Scripts/Global/nph-cc?cc=at HTTP/1.0
```

The first part of the line is the method (GET), the second part is the absolute path and query string, and the last part is the HTTP version. There may be headers generated by the Web browser that sends the request. This request contains no user data.

A common way of generating HTTP requests is by the use of HTML forms. The designer of an HTML form can specify that some of the data entered by the end

user is to be transmitted as user data in the HTTP request. A request generated from a form might therefore include user data as well as the headers described above.

In CICS Web support, the HTTP request is received from TCP/IP for VSE/ESA, and presented to the analyzer, which is a user-replaceable program. The purpose of the analyzer is to decide what CICS resources are needed to satisfy the request. The interpretation of the absolute path as a file reference is not appropriate in the CICS Web support environment, so an enterprise can choose to fix what absolute paths can be sent by browsers, and how the resulting request is interpreted as a request for CICS resources. The functions of the analyzer are described in “Chapter 6. Writing an analyzer for CICS Web support” on page 37. The default analyzer, the URLs that it accepts, and the way it interprets them, are described in “The default analyzer” on page 40.

In the CICS business logic interface, the request for CICS resources is constructed by the caller.

How to receive an HTTP request

There are two ways to receive an HTTP request:

1. Use the EXEC CICS WEB commands (this is the recommended method). See “Using EXEC CICS WEB commands to receive an HTTP request”.
2. Use the environment variables program DFHWBENV (this method is implemented for compatibility with various releases of CICS Transaction Server for OS/390). See “Using DFHWBENV to retrieve information from an HTTP request” on page 71.

Using EXEC CICS WEB commands to receive an HTTP request

When an application receives an HTTP request, the EXTRACT WEB command allows the application processing the request to retrieve information about the inbound request.

This information includes, within the first line of the request, the method to be applied to the resource, the identifier of the resource (URI), the protocol version in use, and any query string supplied on the request.

The EXEC CICS WEB READ/STARTBROWSE/READNEXT/ENDBROWSE HTTPHEADER commands allow the application to extract header information that it wants to read from the HTTP header fields. These headers allow the client to pass on information about the request, and about the client itself, to the server. For example, the user agent indicates the browser being used, and the Content-length gives the length of the body of the HTTP request.

The EXEC CICS WEB READ/STARTBROWSE/READNEXT/ENDBROWSE FORMFIELD commands allow the application to extract name-value pair information from the body of the HTTP request when the body contains HTML forms data. Either URL-encoded or multipart forms data can be used. These commands always return the data in its unescaped form.

Applications that construct symbol lists for the EXEC CICS DOCUMENT API using name-value pairs extracted using the EXEC CICS WEB READ/STARTBROWSE/READNEXT/ENDBROWSE FORMFIELD commands must use the UNESCAPED option on the EXEC CICS DOCUMENT command. If a value contains any ampersands, the application must build the symbol list using a different delimiter byte, and must specify the value of the delimiter on the

DELIMITER option of the EXEC CICS DOCUMENT command. The delimiter chosen must not appear in any of the names or values specified in the symbol list.

The EXEC CICS WEB RECEIVE command allows an application to receive the message body of the HTTP request into a buffer. This means that applications that handle non-forms data, or that prefer to handle the forms data in its escaped form, can then pass the received data unchanged as a symbol list on an EXEC CICS DOCUMENT command. The EXEC CICS WEB RECEIVE command allows the server to receive user data into a buffer and the HTTP Content-length header tells the application the size of the information being sent.

Using DFHWBENV to retrieve information from an HTTP request

You can use the environment variables program DFHWBENV to retrieve the following information present in the HTTP request:

- The IP address of the client
- The IP address of the host
- The local host name
- The HTTP method
- The HTTP version

You can also use the environment variables program to retrieve an indicator of the CICS release under which the program is running. See “Appendix E. Reference information for DFHWBENV” on page 137 for more information about DFHWBENV and the format in which it presents its output.

You can use information from the environment variables program and the information in the communication area to control processing in your CICS program. You should restrict yourself to the DPL subset of the CICS application programming interface. The DPL subset is documented in *CICS Application Programming Reference*.

You can use DFHWBENV in the alias transaction to extract HTTP request header information from the incoming request. Note that you can not invoke DFHWBENV from the analyzer.

Explaining HTTP responses

After receiving and interpreting a request, a server responds with an HTTP response.

An HTTP response consists of a status-line, response header fields and the document data. The status-line contains a numeric status code (STATUSCODE) which defines the response and its associated textual phrase (STATUSTEXT) which gives a short description of the status code. For example:

404 Not Found

This status code indicates that the server has not found anything matching the Request-URI.

See <http://www.w3.org/Protocols/rfc2068/rfc2068> chapter 10 for more information on status codes and reason phrases.

The HTTP response that is sent back to the requester consists of a response line, headers, and optional user data. As in an HTTP request, the CRLF combination

separates the headers, and a null header separates the headers from the user data. A typical response might begin with the response line and the three headers shown:

```
HTTP/1.0 200 Document follows
Date: Fri, 05 Jan 1999 14:23:02 GMT
Server: NCSA/1.5
Content-type: text/html
```

In the first header, HTTP/1.0 is the HTTP version, 200 is the HTTP response code, and Document follows is a user-readable comment. (There are several standard 3-digit response codes; 200 is a response that indicates successful completion of the request.) The next three headers are the date header, the server header, and the content header. The user data might consist of HTML pages, or might be plain text. (In this case the content header promises HTML.)

You can use the EXEC CICS DOCUMENT, EXEC CICS WEB, and EXEC CICS TCP/IP application programming interface to build your response, which is the recommended method, or you can use the HTML template manager DFHWBTL with commarea support, which is implemented for compatibility with various releases of CICS Transaction Server for OS/390.

How to send an HTTP response

There are two ways to construct and send an HTTP response:

1. Use the EXEC CICS application programming interface (this is the recommended method).
2. Use the HTML template manager (this method is implemented for compatibility with various releases of CICS Transaction Server for OS/390).

Using the EXEC CICS API to send an HTTP response

The HTTP header fields allow the server to pass additional information about the response and itself. To add HTTP header information the EXEC CICS WEB WRITE HTTPHEADER command is used. These header fields give information about the server and about further access to the resource identified by the request-URI.

The EXEC CICS WEB SEND command selects a document for delivery. By inserting a document name in the DOCTOKEN option you can specify the name of a document that you wish to send. This document can be a document that has been created using the EXEC CICS DOCUMENT commands. The EXEC CICS WEB RETRIEVE command retrieves a document that has been passed to CICS on an earlier WEB SEND into an application buffer.

The DOCUMENT application programming interface, which is described in the *CICS Enhancements Guide*, allows you to manage CICS documents with the following commands. If you have several different programs building an HTTP response, you can use the combination of EXEC CICS WEB SEND and EXEC CICS WEB RETRIEVE, along with EXEC CICS DOCUMENT CREATE, to pass the partially completed document from one part of the application to the next.

- EXEC CICS DOCUMENT CREATE creates a new document.
- EXEC CICS DOCUMENT RETRIEVE retrieves a copy of the document from the document domain to the application.
- EXEC CICS DOCUMENT INSERT inserts information at a specified point in the document.
- EXEC CICS DOCUMENT SET manipulates symbols and their associated values.

Using the HTML template manager to construct an HTTP response

The HTML template manager DFHWBTL allows you to insert templates in the HTTP response, and to replace symbols in the templates with values that you specify. This has been retained for compatibility with previous releases. See “Appendix D. Reference information for DFHWBTL” on page 131 for more information about the HTML template manager and its operation.

The storage containing the response must begin with a 32-bit integer specifying the length of the response plus 4 for the integer. You can build the HTTP response in the communication area, in which case the maximum length of the response is 4 less than the length of the communication area.

- If your program is operating under CICS Web support or under the CICS business logic interface in *pointer* mode, you can build the response in any area of storage other than the communication area, provided that you pass the address of the storage to **Encode** in the communication area. In this way you can build HTTP responses longer than 32K.
- If your program is operating under the CICS business logic interface in *offset* mode, you can build the response only in the communication area provided.

The response can be constructed entirely by the CICS program, or partly by the CICS program and partly by **Encode**. For commarea-style applications, translation of the various parts of the response from EBCDIC to ASCII (for the headers) and to the client code page (for the user data) is dealt with by the alias program. Web API applications must specify the host and client code pages to be used on the relevant API call.

Handling Escaped Data

The HTTP protocol specifies a set of control characters that are used to define the structure of the stream of data returned in an HTTP response. HTML forms data, for example, uses the “&” character to delimit the end of a name/value pair, so if a user enters an “&” into an HTML form, the HTTP client must send the “&” in a way that does not prevent the HTTP server from correctly parsing the data. The HTTP client does this by “escaping” the character in question. Escaping consists of replacing the relevant character with the string “%nn”, where nn is the ASCII value for the character being unescaped.

Handling escaped data in commarea applications

For commarea-style Web application that have been invoked as a result of an HTTP request being received by a CICS Web TCPIP SERVICE, the way in which CICS handles escaped data depends up the analyzer being used for that TCPIP SERVICE.

On linking to the analyzer program, the HTTP request is in its escaped form. The analyzer can:

- set field WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED, so that the Web application sees the HTTP request in its escaped form.
- leave the data in its unescaped form and ask CICS to unescape the body of the HTTP request by setting WBRA_UNESCAPE to WBRA_UNESCAPE_REQUIRED.
- unescape the HTTP request, then set WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED. This is what the default analyzer DFHWBADX does, to retain compatibility with earlier releases.

The operation of DFHWBUN and DFHWBPA (CICS-supplied utilities to help with the processing of HTTP requests), is affected by whether the data they are processing is escaped or unescaped. CICS uses the setting of WBRA_UNESCAPE to determine this, so you must ensure that on exit from the analyzer URM, WBRA_UNESCAPE is set to WBRA_UNESCAPE_NOT_REQUIRED only if the data is unescaped, otherwise the HTML forms data may not be processed correctly.

If you are writing a commarea-style application that can be run either through CICS Web support or through the CICS business logic interface, you must ensure that WBRA_UNESCAPE is set to WBRA_UNESCAPE_NOT_REQUIRED, and that any escaping is delegated to the Web 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 may cause unpredictable results.

Describing symbols, symbol table, and symbol list

This section describes the symbols in an HTML template, and how the HTML template manager uses the symbol table to replace the symbols with values. The concept of symbol lists and variable substitution is the same for the EXEC CICS WEB application programming interface as for DFHWBTL.

Symbols in an HTML template

In an HTML template, symbols begin with an ampersand (“&”) and end with a semicolon (“;”), and contain up to 32 characters with no imbedded spaces. Thus the following template contains &mytitle; as its only symbol.

```
<html>
  <head>
    <title>
      &mytitle;
    </title>
  </head>
  <body>
```

Symbol lists

This section describes symbol lists as used by the template manager DFHWBTL, which has been implemented for compatibility with various releases of CICS Transaction Server for OS/390.

The template manager maintains a symbol table for each active page environment. In WBTB_BUILD_HTML_PAGE and WBTB_ADD_HTML_TEMPLATE, the template manager uses the input symbol list, if any, to create or update the symbol table, and then replaces the symbols in the template by their values in the table.

A symbol list is a character string. It consists of one or more definitions with single byte separators. By default, the single byte separator is an ampersand, but the caller of the template manager may choose their own separator, as described in “Operational example” on page 75. A definition consists of a name, an equals sign, and a value. Here is an example:

```
mytitle=New Authors&auth1=Halliwell Sutcliffe&auth2=Stanley Weyman
```

The name must contain only uppercase and lowercase letters, numbers, and underscores (“_”). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters. Unlike the symbols in the template, the names in the symbol list have neither an ampersand at the beginning, nor a semicolon at the end. The symbol &mytitle; in the template corresponds to the name mytitle in the symbol list.

The caller of the template manager may specify their own symbol separator to override the default of ampersand. Do this by inserting the character sequence '&DELIM=x&' at the start of the symbol list. The 'x' is a single byte separator used in the following list of symbols. The single byte may be any hexadecimal value apart from:

- null (binary X'00')
- shift in (binary X'0E')
- shift out (binary X'0F')
- space (binary X'40')
- plus sign (binary X'4E')
- percent sign (binary X'6C')
- colon (binary X'7A')
- equals (binary X'7E')
- backslash (binary X'E0')

If the separator is overridden, the application must ensure that the separator does not appear in any symbol value in the symbol list. For this reason, the application should avoid using alphanumeric and other printable characters as the separator.

If any of the above values are specified, they are disregarded, and the template manager assumes that an ampersand is being used as a separator. If a valid separator override is specified, the application must use it to separate symbol values from the symbol names that follow them. The application must also guarantee that the separator does not appear anywhere in a symbol value, and ensure that WBTL_SYMBOL_LIST_LEN includes the length of '&DELIM=x&'.

If an application specifies a valid separator override, all symbol values in the symbol list for this call to the template manager are treated as unescaped character sequences. This means that they are substituted into the template without undergoing any conversion. For example, a plus sign (“+”) remains as a plus, and sequences such as %2B remain as they are rather than being converted to single characters.

If the application does not specify its own valid separator, the following rules apply to symbol values. The value in the symbol list can have any characters except ampersand, but with some restrictions on the use of the percent sign (“%”) and the plus sign (“+”). A percent sign must be followed by two characters that are hexadecimal digits. When the value is put into the symbol table, a plus sign is interpreted as a space, a percent sign and the two hexadecimal digits following it are interpreted as the EBCDIC equivalent of the single ASCII character denoted by the two digits, and the remaining characters are left as they are. If you want a plus sign in the value in the symbol table, you must put %2B; in the value in the symbol list. If you want a percent sign in the value in the symbol table, you must put in the value %25 ; in the symbol list. If you want an ampersand in the value in the symbol table, you must put %26; in the value in the symbol list. If you want a space in the value in the symbol table, the value in your symbol list may contain a space, a plus sign, or %20;.

Operational example

The following symbol list

```
mytitle=New Authors&auth1=Halliwell Sutcliffe&auth2=Stanley Weyman
```

provides definitions of three symbols. Note that an ampersand is a separator that separates a name from the following value, and is not part of the name that follows

it. In an HTML template, &mytitle; is replaced by New Authors, &auth1; by Halliwell Sutcliffe, and &auth2; by Stanley Weyman.

Here is an example of how an application specifies its own separator:

```
&DELIM=!&COMPANY=BLOGGS & SON!ORDER=NUTS + BOLTS
```

Here the symbol COMPANY has a value of 'BLOGGS & SON'. The symbol ORDER has a value of 'NUTS+BOLTS'. The delimiter used is '!', but a non-printable character should be used which will never appear in a symbol value. The use of the UNESCAPED option ensures that the plus sign in 'NUTS+BOLTS' is not converted to a space.

WBTL_SYMBOL_LIST_LEN would be set to decimal 48.

Using the output of the environment variables program

The environment variables program, DFHWBENV, is implemented for compatibility with various releases of CICS Transaction Server for OS/390.

The output of the environment variables program, described in "Appendix E. Reference information for DFHWBENV" on page 137, can be used as a symbol list for the HTML template manager. If you want to use an environment variable that is derived from one of the HTTP headers, you cannot always predict whether it will appear in the environment variables string. Therefore, you should always initialize the symbol table so that names that represent environment variables are associated with default values. Then you can use the output from the environment variables program as a symbol list. For example, if you want to use the &HTTP_REFERER; and &HTTP_AUTHORIZATION; variables in your template, but you do not know whether the client has set them, you could pass the following symbol string to the template manager first:

```
HTTP_REFERER=&HTTP_AUTHORIZATION=
```

This associates both the names with a null string value in the symbol table.

Sample application programs

DFH\$WB1A is a sample program provided with CICS Web support. It uses no converter, and constructs a simple HTTP response whose body is an HTML page. The sample program can be run by enabling CICS Web support with the default analyzer DFHWBADX, and by entering a suitable URL such as the following on a Web browser:

```
http://9.22.123.12:10004/cics/CWBA/DFH$WB1A
```

The format of the URL is described in "Default CICS URL format" on page 79. The response displays the message "DFH\$WB1A on system xxxxxxxx successfully invoked through the CICS Web support." with xxxxxxxx replaced by the application ID of the CICS system in which CICS Web support is running.

DFH\$WB1C is another sample application. It has the same function as DFH\$WB1A, but is written in C.

Sample application DFH0WBCA demonstrates the use of the DOCUMENT API.

Chapter 12. Displaying a template on a Web browser

This chapter contains a simple example of how you could use the WEB and DOCUMENT programming interface and the DOCTEMPLATE and TCPIPSERVICE resource definitions to display a document template on a Web browser. It is supplied in this book as guidance only, and is not intended as comprehensive programming information. Details of the syntax and parameters of the commands used in this example can be found in the *CICS Application Programming Reference*.

In this example:

1. PROGRAM1 displays a template called TEMPLATE1, which invites the user to enter their name.
2. The template specifies PROGRAM2 in its "form action=" field.
3. PROGRAM2 then runs, using as input the user's name from PROGRAM1 and displaying it on the browser as part of a template called TEMPLATE2.

How to display a template on a Web browser

This section provides information about the steps you might follow to display a template:

1. Define and install a TCPIPSERVICE definition to specify the port on which you want CICS and the browser to communicate:

```
TCpipservice ==> MYTCPIP
Group        ==> MYGROUP
DEscription  ==> Provides port number to display template on Web
URM         ==> DFHWBADX
Portnumber   ==> 10004
STatus      ==> Open
TRansaction  ==> CWXN
Backlog     ==> 00001
TSqprefix   ==>
Ippaddress  ==>
```

2. Define and install two DOCTEMPLATE definitions. One specifies a MEMBERNAME of TEMPLAT1 and a TEMPLATENAME=WEBDISPLAY1, as follows:

```
D0ctemplate ==> MYDOCT
Group       ==> MYGROUP
DEscription ==> Template for display on Web
FULL TEMPLATE NAME
TEmplatename ==> WEBDISPLAY1
ASSOCIATED CICS RESOURCE
File        ==>
TSqueue     ==>
TDqueue     ==>
Program     ==>
Exitpgm     ==>
TEMPLATE SUBLIBRARY
Library     ==> DFHHTML
Membername  ==> TEMPLAT1
TEMPLATE PROPERTIES
AppendCrLf ==> YES
TYPe       ==> EbcDic
```

The second specifies a MEMBERNAME of TEMPLAT2 and a TEMPLATENAME of WEBDISPLAY2.

3. Create the first template. In this example, the HTML data is in a VSE/ESA sub-library named DFHDOC in library DFHHTML, in member TEMPLAT1.HTML:

```

<HTML>
<HEAD>
<TITLE>WEB TEMPLATE COMPANY</TITLE>
</HEAD>
<BODY>
<center>
<H2>CICS Web support at work</H2>
<H3>Please enter your first name:</H3>
<FORM METHOD=POST ACTION="http://yoursystem:10004/cics/CWBA/PROGRAM2">
Name:
  <input type=text name=name size=20 maxlength=25>
  <p>
    <input type=submit value="Click here">
</form>
</center>
</BODY>
</HTML>

```

The URL format is described in “Default CICS URL format” on page 79.

4. Create the second template. In this example, the HTML data is in a VSE/ESA sub-library named DFHDOC in library DFHHTML, in member TEMPLAT2.HTML:

```

<HTML>
<HEAD>
<TITLE>WEB TEMPLATE COMPANY</TITLE>
</HEAD>
<BODY>
<center>
<H2>CICS Web support at work</H2>
<H3>Hello &name;, welcome to CICS Web support!</H3>
</center>
</BODY>
</HTML>

```

5. In your PROGRAM1 application program, define a 16–byte field called TOKEN1 to hold the document token, then code the following commands:

```

EXEC CICS DOCUMENT CREATE
      DOCTOKEN(TOKEN1)
      TEMPLATE('WEBDISPLAY1')

```

where the TEMPLATE name is the name specified on the TEMPLATENAME operand of the DEFINE DOCTEMPLATE command. This command creates a document at the location in storage pointed to by TOKEN1. The document contains the HTML in template WEBDISPLAY1.

```

EXEC CICS WEB SEND
      DOCTOKEN(TOKEN1)
      CLNTCODEPAGE('client codepage')

```

This command sends the specified document to a browser. CLNTCODEPAGE can be any of the client codepages listed in “Appendix I. HTML-coded character sets” on page 149.

6. In your PROGRAM2 application program, define an 80–byte buffer, DOCBUF, to hold the retrieved document, a 16–byte field called TOKEN2 to hold the document token, then code the following commands:

```

EXEC CICS WEB RECEIVE
      INTO(DOCBUF)
      LENGTH(DOCLLENGTH)
      MAXLENGTH(80)
      CLNTCODEPAGE('client codepage')
      HOSTCODEPAGE('host codepage')

```

where CLNTCODEPAGE can be any of the client codepages listed in “Appendix I. HTML-coded character sets” on page 149, and HOSTCODEPAGE can be any of the host codepages listed in the *CICS Family: Communicating from CICS on System/390*. The default host codepage is 037.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(TOKEN2)
      TEMPLATE('WEBDISPLAY2')
      SYMBOLLIST(DOCBUF)
      LISTLENGTH(DOCLNGTH)
```

where the TEMPLATE name is the name specified on the TEMPLATENAME operand of the DEFINE DOCTEMPLATE command. This command creates a document at the location in storage pointed to by TOKEN2. The document contains the HTML from template WEBDISPLAY2, with the symbol list retrieved on the WEB RECEIVE command.

```
EXEC CICS WEB SEND
      DOCTOKEN(TOKEN2)
      CLNTCODEPAGE('client codepage')
```

This command sends the specified document to a browser. CLNTCODEPAGE can be any of the client codepages listed in “Appendix I. HTML-coded character sets” on page 149.

7. On your browser, enter a URL like the following:

```
http://yoursystem:10004/cics/CWBA/PROGRAM1
```

The URL format is described in “Default CICS URL format”.

Default CICS URL format

The format of the URL used in this example is the default format used by the analyzer (see “Chapter 6. Writing an analyzer for CICS Web support” on page 37). The format is:

http is the protocol you want the browser to use. This can be **http**.

yoursystem

is the TCP/IP name or IP address for the VSE/ESA system on which CICS is running. If none is specified, this defaults to the IP address of the default TCP/IP stack for the VSE/ESA system on which CICS is running.

10004 is the port number you specified in the TCPIPSERVICE definition. If you are using the HTTP protocol and you omit the port number, it defaults to 80.

cics is the converter name, if you use one, or **cics** if you do not want to use a converter.

CWBA

is the CICS Web transaction.

PROGRAM1

is the name of your program.

Chapter 13. Security for CICS Web support

This chapter discusses security considerations for the HTML template manager VSE/ESA sub-library, and the CICS Web support transactions. It also contains “Using sample programs for security” on page 82 which illustrate the operation of the sample security analyzer, converter, and sign-on program.

Security for the HTML template manager sub-library

If your CICS programs use the VSE/ESA sub-library facilities of the HTML template manager described in “Appendix D. Reference information for DFHWBTL” on page 131, the CICS region user ID must have READ authority for the library described in the DOCTEMPLATE VSE/ESA sub-library definition. If you reference other VSE/ESA sub-libraries by defining DOCTEMPLATES with other libraries, the CICS region must also have READ authority for them.

Security for CICS Web support transactions

You can specify security requirements for each of the transactions that compose the CICS Web support. In the following explanations:

- *authority to attach* means that the associated user must be given READ authority to the named transaction in the resource class specified by the XTRAN system initialization parameter.
- *authority to START* means that the associated user must be given READ authority to the named transaction in the resource class specified by the XPCT system initialization parameter.
- *authority to specify a user ID* means that the associated user must be given READ authority to the `userid.DFHSTART` profile in the SURROGAT resource class, if the XUSER system initialization parameter is specified as YES.
- *authority to use a program* means that the associated user must be given READ authority to the named program in the resource class specified by the XPPT system initialization parameter.

For more information, see the *CICS Security Guide*.

Security for the alias

The alias transaction executes as a non-terminal CICS transaction. Its name is user-specified. If you use the default analyzer described in “The default analyzer” on page 40, the transaction name is the second “index level” in the absolute path specified by the client, and is usually CWBA.

The alias transaction executes under the user ID specified in **wbra_userid**, if it is specified by the analyzer, otherwise it executes under the CICS default userid. If you use the CICS-supplied alias definition, this user ID must have the following authority:

- The authority to attach the alias transaction

If you define your own alias transactions, this user ID must have the following authorities:

- The authority to attach the alias transaction
- The authority to access any CICS resources used by the alias transaction, if it is defined with the RESSEC(YES) option

- The authority to access any CICS system programming commands used by the alias transaction, if it is defined with the CMDSEC(YES) option

Using sample programs for security

If you want a series of Web transactions to be executed under a user ID that is specified by the Web client (the end user), you can use the security sample programs to help you. To use the security analyzer sample program, you must specify its name as the Analyzer Program name in the TCPIPSERVICE definition.

Two sets of sample programs are provided:

1. The security sample programs, described in “The security sample programs”:
 - The security analyzer, DFH\$WBSA
 - The security converter, DFH\$WBSC
 - The sign-on program, DFH\$WBSN
2. The basic authentication sample programs, described in “Using basic authentication sample programs” on page 83:
 - The basic authentication analyzer, DFH\$WBAU
 - The basic authentication converter, DFH\$WBSB

The CICS resource definitions for these programs are in group DFH\$WBSN.

The security sample programs

The security sample programs use the state management sample program, DFH\$WBST.

A typical sequence of interactions between a user and the CICS Web support might be as follows:

1. The end user sends an HTTP request in which the URL has no query string.
2. The security analyzer checks the URL for a converter name, alias name, program name, and query string. As there is no query string, it sets its outputs so that the converter is the security converter sample program DFH\$WBSC, while the alias and CICS program are the ones requested in the URL. The user token output is zeros.
3. The **Decode** function of the security converter, finding a zero user token, calls the Create function of the state management sample program to assign a token. It saves the token in its user token output. It uses the Store function of the state management program to save the original URL. It sets the CICS program name to DFH\$WBSN, the security sign-on sample program.
4. The sign-on program builds an HTML form asking for a user ID and a password. The form specifies an HTML ACTION that generates a URL. The generated URL causes the sign-on program to be invoked again, but with the state management token as its query string.
5. The **Encode** function of the security converter builds the HTTP response.
6. The user gets the form, fills in the user ID and the password, and sends it back.
7. The security analyzer finds a query string. It uses the Retrieve function of the state management program to validate the token. As the token is not yet associated with a valid user ID, it sets its outputs so that the converter name is the security converter. The state management token is passed as the user token.

8. The sign-on program extracts the user ID and password from the form, and uses EXEC CICS VERIFY PASSWORD to validate the user ID. DFH\$WBSN passes the validated user ID in the commarea to the security converter (DFH\$WBSC), which uses the Store function of the state management program.
9. The **Encode** function of the security converter builds the HTTP response, and adds a redirection (HTTP response 302) to it, incorporating the original URL.
10. When the user has entered a valid userid and password, CICS issues an HTTP Redirect response containing the original URL, with the security token appended to it. When the browser resubmits the redirected request, CICS knows that signon processing is complete, and processes the request.
11. The Web browser receives the redirected URL, and sends a request for the original program with the token that identifies the validated user ID.
12. The security analyzer finds that the query string is a valid user token associated with a user ID, so the original request proceeds.

Once the user token has been established as the key to the authenticated user ID, it is the responsibility of the CICS program, or the converter that builds the HTTP response, to ensure that any URLs that are generated to continue the conversation with the client contain the conversation token as query string. This ensures that subsequent programs in the conversation execute under the specified user ID. Since the CICS program is already running with the correct conversation token as its query string, it can extract its value by using the environment variable program to obtain the value of the query string. If necessary, the correct value for the conversation token can be substituted into HTML templates by using the symbol &QUERY_STRING;, provided that the environment variable string has first been loaded into the symbol table in the template manager's page environment.

Using basic authentication sample programs

The basic authentication sample programs use HTTP basic authentication. On the first reference by a Web browser to a CICS region (identified by its application ID), the browser prompts the user for a user ID and password. The user ID and password supplied at the prompt are sent to the CICS region for every request. CICS validates the user ID and password for each request. There is no user prompt for the second or later requests.

The user ID and password are encoded, but not encrypted, for transmission.

To use the security analyzer sample program, you must specify its name as the TCPIPSERVICE definition.

The basic authentication analyzer searches the incoming HTTP headers for an Authorization header with a "Basic" operand. If it finds one, it decodes the BASE64-encoded user ID and uses it as the alias user ID. It always schedules DFH\$WBAU as the converter.

The basic authentication converter searches the incoming HTTP headers for an Authorization header. It decodes the user ID and the password. It uses VERIFY PASSWORD to validate the password. If the user ID and password combination is invalid, or if the Authorization header is absent, an HTTP 401 response is returned to the Web browser, and the user is prompted for a password. If the user ID and password combination is correct, the application continues, and runs under the specified user ID.

Chapter 14. Problem determination

This chapter contains Diagnosis, Modification, or Tuning Information.

This chapter helps you debug problems in CICS Web support and CICS business logic interface user-replaceable programs, the IBM-supplied parts of CICS Web support and CICS business logic interface, and the operating environment of CICS Web support. If you suspect you have a problem in another part of CICS, refer to the *CICS Problem Determination Guide*.

The formats of messages and trace outputs in CICS Web support and CICS business logic interface are also described.

Diagnostic information is designed to provide first failure data capture, so that if an error occurs, enough information about the error is available directly without the need to reproduce the error situation. The information is presented in the following forms:

Messages

CICS Web support and CICS business logic interface provide CICS messages with the prefix DFHWB, and these are listed in the *CICS Messages and Codes* manual.

Trace CICS Web support and CICS business logic interface output system trace entries containing all the important information required for problem diagnosis.

Dump Dump formatting is provided for data areas relating to CICS Web support and CICS business logic interface.

Abend codes

Transaction abend codes are standard four-character names. The abend codes are listed in the *CICS Messages and Codes* manual.

This chapter is organized as follows:

- “Recovery procedures (CICS Web support)” on page 86 describes how CICS Web support copes with software errors.
- “Product design considerations (CICS Web support)” on page 86 describes aspects of the design of CICS Web support that you need to know for problem determination.
- “Troubleshooting” on page 86 describes a method of analyzing problems in CICS Web support and CICS business logic interface.
- “Using messages and codes” on page 87 describes how to find information about messages and abend codes.
- “CICS Web support and CICS business logic interface trace information” on page 87 describes CICS Web support and CICS business logic interface trace information.
- “Dump and trace formatting” on page 88 describes how to control the formatting of dumps and trace entries.
- “Debugging the user-replaceable programs” on page 89 gives hints about debugging user-replaceable programs.

Recovery procedures (CICS Web support)

If a TCPIP SERVICE definition is installed and enabled when CICS fails, that definition is re-installed and re-enabled when CICS recovers. Any changes made to the TCPIP SERVICE with CEMT are not recovered.

If TCP/IP for VSE/ESA abends, CICS Web support enters immediate disable processing, but CICS continues to run.

The abending of an alias transaction might cause changes to recoverable resources to be backed out.

Product design considerations (CICS Web support)

There are two CICS transactions for each HTTP request; CWXN (or an alias of CWXN), and CWBA (or an alias of CWBA). These two transactions have different logical units of work.

Troubleshooting

This section provides some hints on troubleshooting. It follows the general outline:

1. Define the problem.
2. Obtain information (documentation) on the problem.
3. Work out where in CICS Web support the problem is happening.

Defining the problem

When you have a problem, first try to define the circumstances that gave rise to it. If you need to report the problem to the IBM software support center, this information is useful to the support personnel.

1. What is the system configuration?
 - CICS Transaction Server for VSE/ESA release
 - VSE/ESA operating system
 - Language Environment release
2. What operating options are in use?
3. When did the problem first occur?
4. What were you trying to accomplish at the time the problem occurred?
5. What changes were made to the system before the occurrence of the problem?
 - To the VSE/ESA system
 - To CICS Web support
 - To the CICS program being called by the client
 - To the converter being used in the call
 - To the analyzer being used to interpret client requests
 - To the client
 - To CICS Transaction Server for VSE/ESA
 - To TCP/IP for VSE/ESA
6. What is the problem?
 - Incorrect output
 - Hang/Wait: Use CEMT INQUIRE to display details of the transaction
 - Loop: Use CEMT INQUIRE to display the details of the transaction
 - Abend in a user-replaceable program
 - Abend in a CICS program
 - Abend in the IBM-supplied part of CICS Web support
 - Performance problem
 - Storage violation

- Logic error
7. At what point in the processing did the problem occur? (Use Figure 8 on page 19.)
 8. What was the state of TCP/IP for VSE/ESA

Documentation about the problem

To investigate most problems, you must look at the dumps, traces, and logs provided with VSE/ESA and CICS.

- System dump: This contains the CICS internal trace
- CICS auxiliary trace, if enabled
- Console log
- CSMT log
- CWBO log
- CICS job log

To identify which are likely to be useful for your problem, try to work out the area of the CICS Web support giving rise to the problem, and read the relevant section in the rest of this chapter.

Using messages and codes

CICS Web support and CICS business logic interface messages have identifiers of the form DFHxxnnnn., where *nnnn* are four numeric characters indicating which component generated the message, as shown in *CICS Messages and Codes*. *xx* indicates which domain generated the message; WB indicates the Web domain, DH indicates the document handler domain, and SO indicates the Sockets domain.

CICS Web support messages are sent to the CICS Web support message transient data queue CWBO. CICS Sockets domain messages are sent to the CICS Sockets domain message transient data queue CSOO. If you define CWBO as an indirect destination for CSMT, the messages appear in CSMT. Some messages are sent to the console.

When the CICS Web support or the CICS business logic interface issues a message as a result of an error, it also makes an exception trace entry. The CICS Web support also generates information messages, for instance during enable processing and disable processing.

Messages are supplied in English, Japanese, Chinese, Korean and German.

The CICS Web support and CICS business logic interface abend codes are listed in *CICS Messages and Codes*.

CICS Web support and CICS business logic interface trace information

The CICS Web support and CICS business logic interface output CICS system trace, which is formatted using software supplied as part of CICS.

If selected, level 2 trace gives a full trace of the data being transmitted between the client and the CICS program. CICS trace output is described in the *CICS Problem Determination Guide*, and details of the contents of each trace point are given in the *CICS User's Handbook*.

Numeric values of symbolic codes

The response codes from the analyzer and the converter appear in the trace output as numeric values as follows:

- URP_OK (0)
- URP_EXCEPTION (4)
- URP_INVALID (8)
- URP_DISASTER (12)
- URP_OK_LOOP (16)

The CICS-defined reason codes from the analyzer and converter appear in the trace output as numeric codes as follows:

- URP_SECURITY_FAILURE (1)
- URP_CORRUPT_CLIENT_DATA (2)

The DFHWBTL function codes appear in the trace output as numeric values as follows:

- WBTL_BUILD_HTML_PAGE (1)
- WBTL_START_HTML_PAGE (2)
- WBTL_ADD_HTML_SYMBOLS (3)
- WBTL_ADD_HTML_TEMPLATE (5)
- WBTL_END_HTML_PAGE (6)

The response codes from DFHWBTL appear in the trace output as numeric values as follows:

- WBTL_OK (0)
- WBTL_EXCEPTION (4)
- WBTL_INVALID (8)
- WBTL_DISASTER (12)

The reason codes from DFHWBTL appear in the trace output as numeric values as follows:

- WBTL_INVALID_FUNCTION (1)
- WBTL_INVALID_TOKEN (2)
- WBTL_INVALID_SYMBOL_LIST (3)
- WBTL_INVALID_BUFFER_PTR (4)
- WBTL_FEATURE_INACTIVE (5)
- WBTL_TEMPLATE_NOT_FOUND (6)
- WBTL_TEMPLATE_TRUNCATED (7)
- WBTL_PAGE_TRUNCATED (8)
- WBTL_GETMAIN_ERROR (9)
- WBTL_FREEMAIN_ERROR (10)

Dump and trace formatting

To select the level of dump formatting printed for the CICS Web support or CICS business logic interface, you change the CICS parameters in the INFOANA control statement for dump formatting as follows:

```
CALL DFHPD410 DATA WB=0|1|2|3,TR=1|2|3
```

The parameters have these meanings:

WB=0 Suppress system dumps for the CICS Web support, and the CICS business logic interface.

WB=1 Produce a system dump summary listing for the CICS Web support, and the CICS business logic interface.

WB=2 Produce a system dump for the CICS Web support, and the CICS business logic interface.

WB=3 Produce a system dump summary listing and a system dump for the CICS Web support, and the CICS business logic interface.

TR=1 Produce an abbreviated trace.

TR=2 Produce a full trace.

TR=3 Produce both a full and abbreviated trace.

CICS Web support output in the formatted dump consists of the major CICS Web support control blocks, with interpretation of some of the fields.

The CICS Web support output can be found in the INFOANA output by searching for ==WB. It is under the heading CICS Web support.

The CICS Sockets output can be found in the INFOANA output by searching for ==S0. It is under the heading CICS Sockets.

The document domain output can be found in the INFOANA output by searching for ==DH. It is under the heading Document domain.

Debugging the user-replaceable programs

The user-replaceable programs are the analyzer (CICS Web support only) and the converters

Using EDF

You can use EDF with the analyzer, and you can also use it to debug the converter, and the CICS program. If you want to use EDF, you must:

- Define EDF as a translator option when the program is translated.
- Define CEDF(YES) in the program definition of the converter, the CICS program, or the analyzer.
- Enter CEDX xxxx at the terminal, where xxxx is either CWXN or an alias of CWXN (to debug the analyzer), or CWBA or an alias of CWBA (to debug the converter or the target program).

Using trace entries

You can output diagnostic information to the CICS trace by the use of the EXEC CICS ENTER TRACENUM command. The amount of trace information and the information contained within trace entries is at your discretion. See the *CICS Application Programming Reference* for more information about this command.

Writing messages

You can write diagnostic messages by using EXEC CICS WRITEQ TD. Message information content, message format, frequency, and destination are at your discretion.

Trapping abends

You are recommended to use EXEC CICS HANDLE ABEND to trap abends. You should collect the diagnostic information you need by tracing, and then return a URP_DISASTER response.

Part 3. The CICS business logic interface

This part of the book contains information about the CICS business logic interface.

This part of the book describes the CICS business logic interface. CICS Web support, as described in “Chapter 3. Introduction to CICS Web support” on page 17, is a collection of CICS resources supporting direct access to CICS transaction processing services from Web browsers. The CICS business logic interface is a callable program that allows a variety of callers to access the same Web-aware business logic as CICS Web support, but via a CICS link rather than via the CICS HTTP listener.

The CICS business logic interface supports the separation of presentation logic from business logic in application design. The converter program contains the presentation logic and understands how data must be presented to the business logic, which is contained in the application program. There is a brief discussion about the distinction between presentation logic and business logic in “Separating business and presentation logic” on page 11.

It contains:

- “Chapter 15. Introduction to the CICS business logic interface” on page 93
- “Chapter 16. Configuring the CICS business logic interface” on page 101

Chapter 15. Introduction to the CICS business logic interface

This chapter presents an overview of the CICS business logic interface. It contains the following sections:

- “Types of requester”
- “Processing examples”
- “Controlling flow in request processing” on page 94
- “Passing data in request processing” on page 96

“Chapter 16. Configuring the CICS business logic interface” on page 101 provides information about setting up the CICS business logic interface.

Types of requester

The CICS business logic interface can deal with requests from the following types of requester. These callers provide a communication area that contains parameters that specify the required CICS transaction processing services. For example:

- Users of the external CICS interface (EXCI).
- Users of the CICS Family: Client/Server Programming external call interface (ECI).
- Any program running in a CICS application environment. The program uses EXEC CICS LINK to the CICS business logic interface.

Processing examples

Figure 12 shows how the CICS business logic interface processes a request from a VSE/ESA application that uses the EXCI.

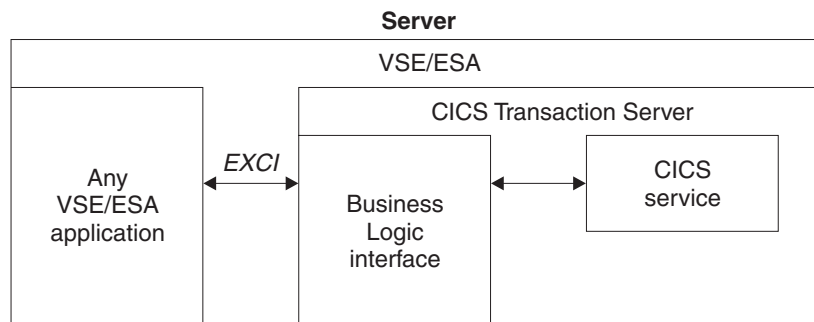


Figure 12. Processing a request from the EXCI

The VSE/ESA application constructs a communication area that contains parameters for the CICS business logic interface, and calls it with EXCI. The CICS business logic interface ensures that the CICS Transaction Server for VSE/ESA provides the requested service, and returns any output in the communication area.

Figure 13 on page 94 shows how the CICS business logic interface processes a request from a CICS client that uses the ECI.

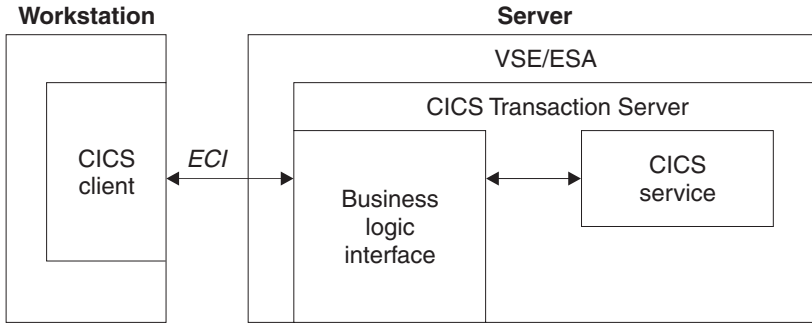


Figure 13. Processing a request from the ECI

The client, running in a workstation environment, constructs a communication area that contains parameters for the CICS business logic interface. It uses the ECI to call the CICS business logic interface. The CICS business logic interface ensures that the CICS Transaction Server for VSE/ESA provides the requested service, and returns any output in the communication area. The ECI operates with the SNA protocol (see the *CICS Family: Client/Server Programming* for further information).

Controlling 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 14 shows the control flow through the CICS business logic interface to a program. The CICS business logic interface is accessed by an EXEC CICS LINK command to PROGRAM DFHWBBLI.

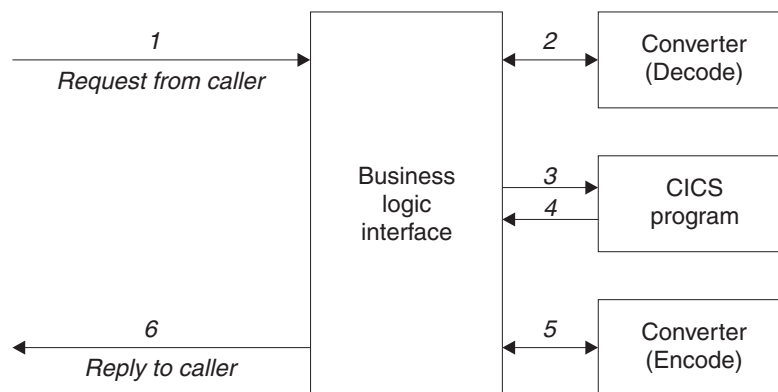


Figure 14. Calling a program with the CICS business logic interface—control flow

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 communication area for the CICS program.
3. The CICS business logic interface calls the CICS program that the caller specified. The communication area passed to the CICS program is the one set up by **Decode**. If no converter program was called, the communication area contains the entire request.

4. The CICS program processes the request, and returns output in the communication area.
5. If the caller requested a converter, the CICS business logic interface calls the **Encode** function of the converter, which uses the communication area to prepare the response. If no converter program was called, the CICS business logic interface assumes that the CICS program has put the desired response in the communication area.
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 15 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 DFHWTBTA as the CICS program to be called, in `wbbl_server_program_name`.

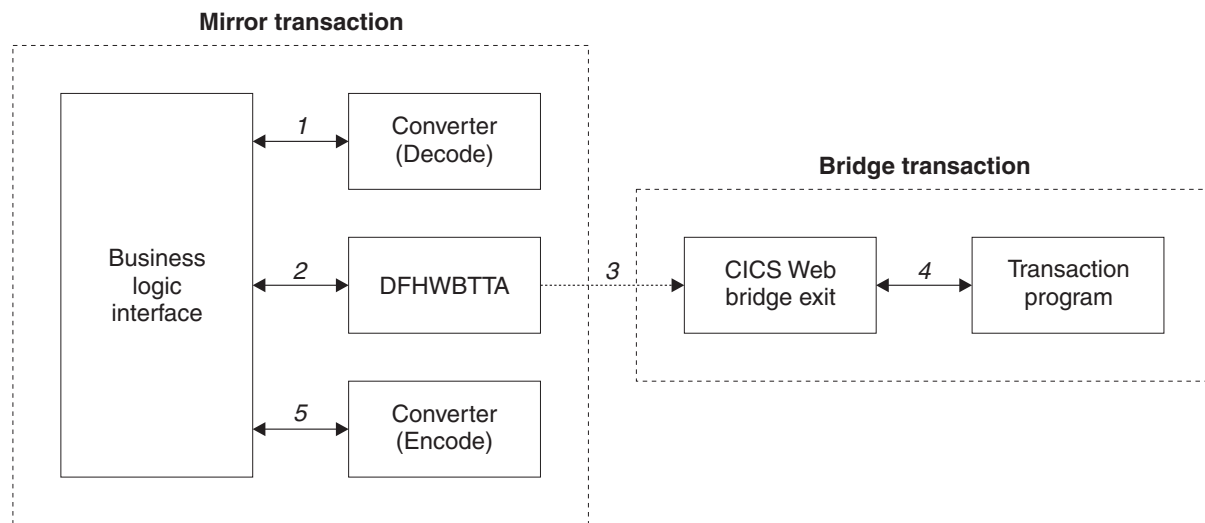


Figure 15. Running a transaction with the CICS business logic interface—control flow

1. If the caller requests a converter, the CICS business logic interface calls it, requesting the **Decode** function. **Decode** sets up the communication area for DFHWTBTA.
2. The CICS business logic interface calls DFHWTBTA. The communication area passed to DFHWTBTA is the one set up by **Decode**. If no converter program was called, the communication area contains the entire request.
3. DFHWTBTA 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, and 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 communication area to prepare the response. If no converter program was called, the CICS business logic interface assumes that the communication area contains the desired response.

Passing data 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.

Using the CICS business logic interface to call a program

Figure 16 shows the data flow through the CICS business logic interface to a program, and back to the requester.

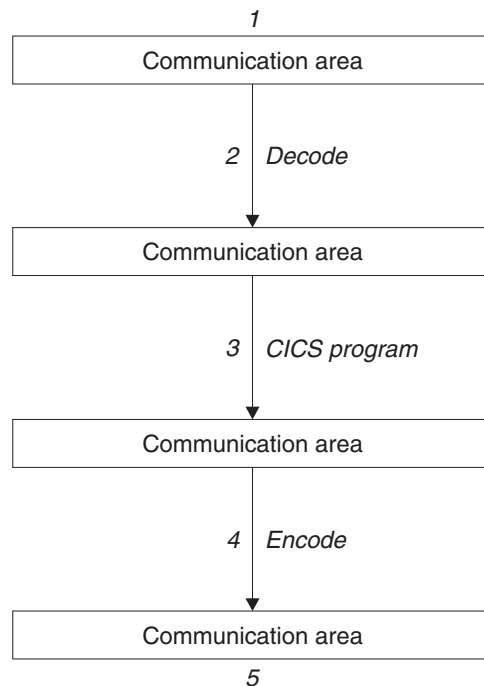


Figure 16. Calling a program with the CICS business logic interface—data flow

1. The caller of the CICS business logic interface provides a communication area that contains the request to be processed. The contents of the communication area 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 communication area for the CICS program.
3. The CICS program updates the communication area.
4. If the caller requests a converter, the **Encode** function of the converter constructs the communication area 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 communication area.

Requesting a terminal-oriented transaction

Figure 17 on page 97 shows the data flow for a request that starts a terminal-oriented transaction.

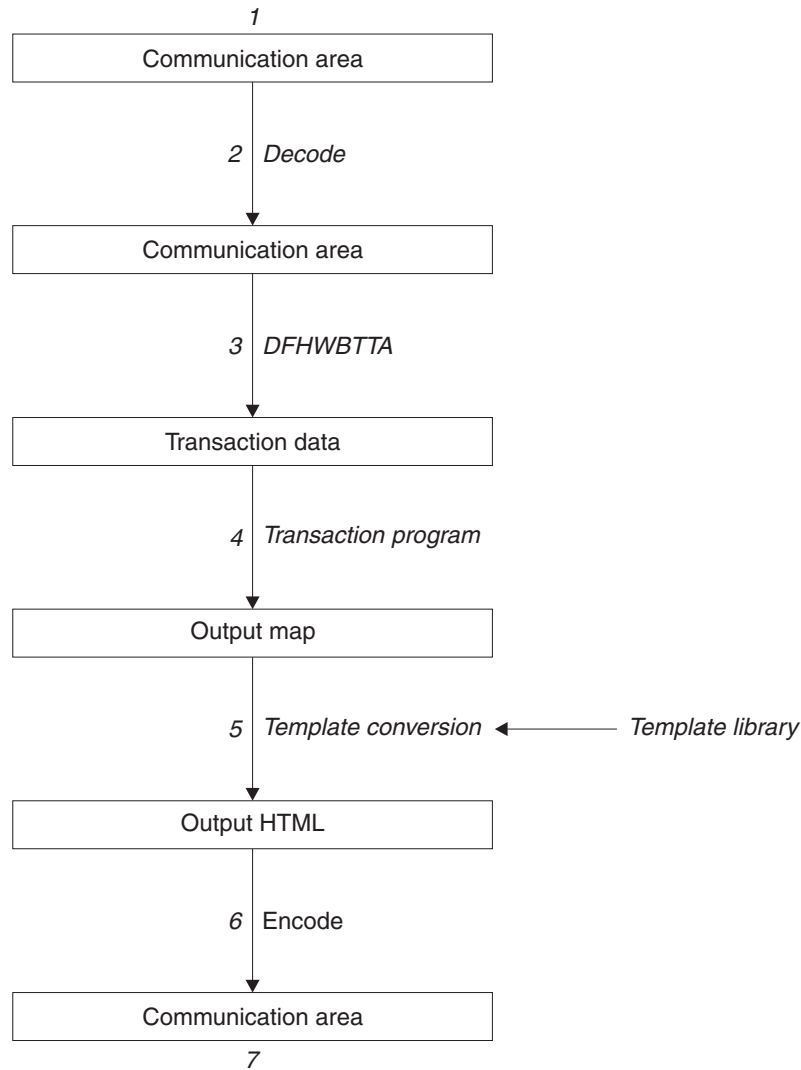


Figure 17. Starting a terminal-oriented transaction—data flow

This figure shows the data flow through the CICS business logic interface for a 3270 BMS application. If CICS Web support processes the request, there is data conversion of headers and user data as shown in Figure 10 on page 22.

1. The caller of the CICS business logic interface provides a communication area that contains the request to be processed. The contents of the communication area 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 EXEC CICS RECEIVE.
4. The transaction program uses EXEC CICS RECEIVE to receive the data. It then constructs an output map, and uses EXEC CICS SEND MAP 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 communication area.

Figure 18 shows the data flow for a request that continues a terminal-oriented transaction.

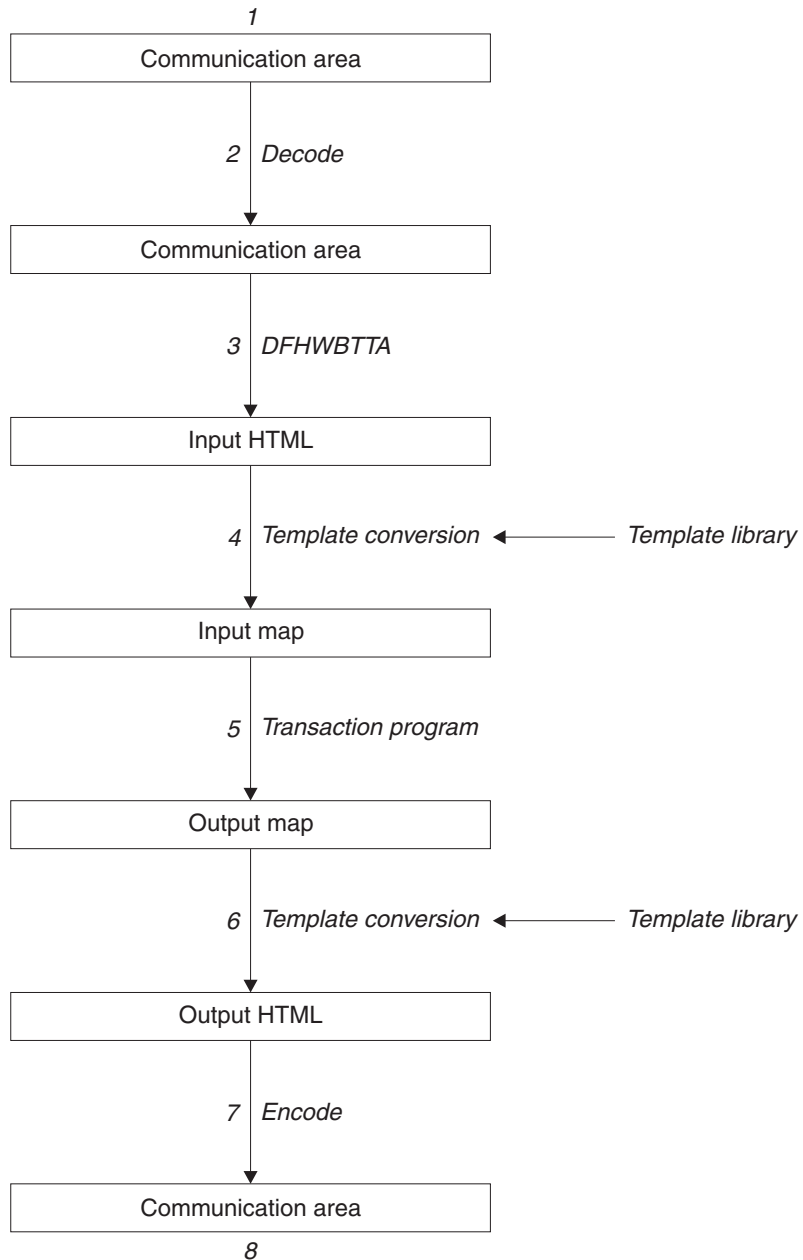


Figure 18. Continuing a terminal-oriented transaction—data flow

This figure shows the data flow when the CICS business logic interface processes the request. If CICS Web support processes the request, there is data conversion of headers and user data as shown in Figure 10 on page 22.

1. The caller of the CICS business logic interface provides a communication area that contains the request to be processed. The contents of the communication

area 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 communication area 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 EXEC CICS RECEIVE MAP.
4. The incoming forms input data is converted into a BMS map. This conversion uses templates from DOCTEMPLATE definitions.
5. The transaction program uses EXEC CICS RECEIVE MAP to receive the data. It then constructs an output map, and uses EXEC CICS SEND MAP 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 communication area to be returned to the caller.
8. The CICS business logic interface returns to its caller, which can now use the contents of the communication area.

Chapter 16. Configuring the CICS business logic interface

The CICS business logic interface is a callable program that does not require the support of special transactions. However, before you plan how to use the CICS business logic interface, you need to know about the role of the converters.

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 14 on page 94 and Figure 15 on page 95. Each converter must provide two functions:

- **Decode** is used before the CICS program is called. It can:
 - Use the data from the incoming request to build the communication area in the format expected by the CICS program.
 - Supply the lengths of the input and output data in the CICS program communication area.
 - Perform administrative tasks related to the request.
- **Encode** is used after the CICS program has been called. It can:
 - Use the data from the CICS program to build the response.
 - Perform administrative tasks related to the response.

There are some restrictions on the functions of the converter that depend on how the CICS business logic interface was called. The two modes of calling the CICS business logic interface are:

1. Pointer mode
2. Offset mode

The differences in the functions are described in “Chapter 7. Writing a converter” on page 43, and in “Appendix A. Reference information for DFHWBBLI” on page 107. Converters called in offset mode are more restricted than converters called in pointer mode. All requests from web browsers using the CICS Transaction Gateway result in offset mode calls to the CICS business logic interface.

You must set the WEBDELAY system initialization parameter, as described in “Controlling web support with system initialization parameters” on page 29.

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.

Reference information for the business logic interface can be found in “Appendix A. Reference information for DFHWBBLI” on page 107

Chapter 17. Programming tasks for client systems

- Write VSE applications to use the EXCI to communicate with the CICS business logic interface. There are applications that use CICS programs for their services, and applications that use CICS transactions for their services. See “Appendix A. Reference information for DFHWBBLI” on page 107.
- Write workstation applications to use the ECI to communicate with the CICS business logic interface. There are applications that use CICS programs for their services, and applications that use CICS transactions for their services. See “Appendix A. Reference information for DFHWBBLI” on page 107.
- Write CICS applications to use EXEC CICS LINK to communicate with the CICS business logic interface. There are applications that use CICS programs for their services, and applications that use CICS transactions for their services. See “Appendix A. Reference information for DFHWBBLI” on page 107.

Part 4. Appendixes

Appendix A. Reference information for DFHWBBLI

This section contains Product-sensitive Programming Interface and Associated Guidance Information. It provides reference information for the business logic interface.

Business logic interface

Summary of parameters

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 the CICS Web support. The files for the various languages are as follows:

Language	File
Assembler	DFHWBBLD
C	DFHWBBLH
COBOL	DFHWBBLO
PL/I	DFHWBBLI

These files give language-specific information about the data types of the fields in the communication area.

In the following table, the names of the parameters are given in abbreviated form: each name in the table must be prefixed with **wbbl_** to give the name of the parameter.

Table 2. Parameters for the business logic interface

Input wbbl_	Inout wbbl_	Output wbbl_
client_address client_address_length client_address_string converter_program_name eyecatcher header_length header_offset http_version_length http_version_offset indata_length indata_offset indata_ptr length method_length method_offset mode prolog_size resource_length resource_offset server_program_name status_size user_token user_data_length vector_size version		outdata_length outdata_offset outdata_ptr

Function

The business logic interface allows callers to specify what presentation logic is to be executed before and after a CICS program. It has two modes of operation:

- Pointer mode: the input data for **Decode** is in storage allocated separately from the communication area for the business logic interface. The communication area 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 communication area for the business logic interface, and the communication area contains a pointer (**wbbl_outdata_ptr**) to the output from **Encode**.
- Offset mode: the input data for **Decode** is part of the communication area for the business logic interface. The communication area 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 communication area for the business logic interface, and the communication area 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 “Chapter 7. Writing a converter” on page 43.

Note: The business logic interface does not handle the response codes and reason codes produced by the converter in the manner described in “Appendix C. Reference information for the converter” on page 121, but as described in “Responses” on page 112 under responses 400, 500, and 501.

Parameters

Before inserting the inputs into the communication area, you must clear it to binary zeros.

wbbl_eyecatcher

(Input only)

A 14-character field that must be set to the standard eyecatcher string **>DFHWBBLIPARMS**.

wbbl_client_address

(Input only)

A fullword 32-bit field that must be set to the binary IP address of the client, if this is known.

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_address_string

(Input only)

A string of up to 15 characters that are the dotted decimal representation `wbbl_client_address`, padded on the right with binary zeros.

wbbl_converter_program_name

(Input only)

The 8-character name of the program to be used to 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 this 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", this field 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 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 should be 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 should be 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", 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 is not necessarily the same as `wbbl_indata_offset`.

wbbl_outdata_ptr

(Input only)

If `wbbl_mode` is "P", this is the fullword address in which DFHWBBLI returns the address of the response data from the application. This address is 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 slash (/) 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 slash (/) in the URL).

wbbl_response

(Input only)

A fullword binary field in which DFHWBBLI returns its response code.

wbbl_server_program_name

(Input only)

The 8-character name of the application program that is to be used to process the request and produce the response.

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 user data. If the analyzer modified this 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 should be set using the constant value `wbbl_current_version`.

Responses

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** The EXEC CICS LINK 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** The EXEC CICS LINK 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.
 - The EXEC CICS LINK 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:

- The EXEC CICS LINK 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).
- The EXEC CICS LINK 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 B. Reference information for DFHWBADX

This section contains Product-sensitive Programming Interface and Associated Guidance Information. It provides reference information for the analyzer, and information about the responses and reason codes for the default analyzer, DFHWBADX.

Summary of parameters

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 the CICS Web support. The files for the various languages are listed in the following table.

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 communication area. If you use these files you must specify XOPTS(NOLINKAGE) on the Translator step; failure to do this causes the compile to fail.

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 3. Parameters for the analyzer

Input wbra_	Inout wbra_	Output wbra_
client_ip_address content_length eyecatcher function http_version_length http_version_ptr method_length method_ptr request_header_length request_header_ptr request_type resource_length resource_ptr server_ip_address user_data_ptr	user_data_length userid	alias_tranid converter_program dfhcnv_key reason response server_program user_token unescape

Function

The analyzer is called by Web attach processing before it starts the alias. The analyzer can examine the incoming request, and must specify the CICS resources needed to process the request.

Parameters

wbra_alias_tranid

(Output only)

A string of length 4. The transaction ID of the alias that is to service the request. If you do not set this field, or if you set it to blanks, CWBA is used.

wbra_client_ip_address

(Input only)

The 32-bit IP address of the client.

wbra_content_length

(Input only)

A 32-bit binary representation of the user data length as specified by the Content-Length HTTP header in the received data.

wbra_converter_program

(Output only)

A string of length 8. The name of the converter whose **Decode** and **Encode** functions are used to process the request. If you do not set this field, no converter is called.

wbra_dfhcnv_key

(Output only)

A string of length 8. The name of the conversion template in the DFHCNV table for the code page translation of the user data for this request. If the request is not an HTTP request, this name is used to translate the entire request. The name you choose must be defined in the DFHCNV table, as described in "Defining a conversion table" on page 33. If you do not set this field, there is no translation.

wbra_eyecatcher

(Input only)

A string of length 8. Its value is ">analyze".

wbra_function

(Input only)

A code indicating that the analyzer is being called. The value is 1.

wbra_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, this length is zero.

wbra_http_version_ptr

(Input only)

A pointer to the string identifying the HTTP version supported by the client. If the request is not an HTTP request, do not use this pointer.

wbra_method_length

(Input only)

The length in bytes of the string identifying the method specified in the HTTP request. If the request is not an HTTP request, this length is zero.

wbra_method_ptr

(Input only)

A pointer to the method specified in the HTTP request. If the request is not an HTTP request, do not use this pointer.

wbra_reason

(Output only)

A reason code—see “Responses and reason codes” on page 118.

wbra_request_header_length

(Input only)

The length of the first HTTP header in the HTTP request. If the request is not an HTTP request, this length is zero.

wbra_request_header_ptr

(Input only)

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, do not use this pointer.

wbra_request_type

(Input only)

If this is an HTTP request, the value is `WBRA_REQUEST_HTTP`. If this is not an HTTP request, the value is `WBRA_REQUEST_NON_HTTP`.

wbra_resource_length

(Input only)

The length in bytes of the string identifying the HTTP absolute path specified in the HTTP request. If the request is not an HTTP request, this length is zero.

wbra_resource_ptr

(Input only)

A pointer to the string identifying the HTTP absolute path specified in the HTTP request. If the request is not an HTTP request, do not use this pointer.

wbra_response

(Output only)

A response—see “Responses and reason codes” on page 118.

wbra_server_ip_address

(Input only)

The 32-bit IP address of the TCP/IP for VSE/ESA partition receiving the request.

wbra_server_program

(Output only)

A string of length 8. The name that is passed to **Decode** as **decode_server_program**. If you do not set this field, the value passed is nulls. The program name must be set here or in the **Decode** function of the converter specified in **wbra_converter_program**, or no CICS program will be called.

wbra_unescape

(Output only)

The default CICS action for escaped HTTP data is to pass the data to the application in its escaped form. To ensure that escaped characters are

unescaped before passing them to your application program, the value is `WBRA_UNESCAPE_REQUIRED`; otherwise the value is `WBRA_UNESCAPE_NOT_REQUIRED`.

wbra_user_data_length

(Input and output)

A 15-bit integer, representing the length of the user data in the HTTP request. If the request is non-HTTP, this length is the length of the request. The length passed to the analyzer includes any trailing carriage return and line feed (CRLF) characters that may delimit the end of the user data. If the length is reduced, the newly redundant bytes are replaced by null characters, `X'00'`. The modified value is passed on to the CICS business logic interface in field `wbbi_user_data_length`, and to the **Decode** program in field `decode_user_data_length`.

wbra_user_data_ptr

(Input only)

A pointer to the user data in the HTTP request. If the request is not an HTTP request, this is a pointer to the request.

wbra_user_token

(Output only)

A 64-bit token that is passed to **Decode** as `decode_user_token`. If you do not set this field, the value passed is null. If there is no converter for this request, the value is ignored.

wbra_userid

(Input and output)

A string of length 8. On input, it is the userid derived from the client certificate, if one was used. On output, it is the userid under which the alias executes. If it is blank or null on output, the CICS default userid is used.

Responses and reason codes

You must return one of the following values in `wbra_response`:

URP_OK

Web attach processing starts the alias transaction.

URP_EXCEPTION

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, response 400 is sent to the Web browser.

If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

URP_INVALID

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, response 400 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

URP_DISASTER

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, response 400 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If you return any other value in **wbra_response**, Web attach processing writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, response 400 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

You may supply a 32-bit reason code in **wbra_reason** to provide further information in error cases. The CICS Web support does not take any action on the reason code returned by the analyzer. The reason code is output in any trace entry that results from the invocation of the analyzer, and in message DFHWB0523.

See “Numeric values of symbolic codes” on page 88 for the numeric values of the response and reason codes in trace output.

DFHWBADX responses and reason codes

The meanings of the responses produced by the default analyzer 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. (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 “/”.
- 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 program name in the resource specification was 0 or more than 8.
- 8 One of the following:
 - There is no second “/”
 - There is nothing after the second “/”
 - There is no third “/”
 - There is nothing after the third “/”

URP_INVALID

The eye-catcher was invalid. This is an internal error.

Appendix C. Reference information for the converter

This section provides reference information for the **Decode** and **Encode** functions of the converter.

The names of the parameters and constants in the communication area passed to the converter, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of the 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 communication area. If you use these files you must specify XOPTS(NOLINKAGE) on the Translator step; failure to do this causes the compile to fail.

Decode

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 4. Parameters for **Decode**

Input decode_	Inout decode_	Output decode_
client_address client_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	data_ptr input_data_len server_program user_token	output_data_len reason response

Function

If the analyzer, or the caller of the CICS business logic interface, specified a converter name for the request, **Decode** is called before the CICS program that is to service the request.

Parameters

decode_client_address

(Input only)

The 32-bit IP address of the client.

decode_client_address_string

(Input only)

The IP address of the client in dotted decimal format.

decode_data_ptr

(Input and output)

On input, a pointer to the request from the client (as modified by the analyzer) 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 communication area to be passed to the CICS program. You must ensure that the pointer points to a valid location, or results can be unpredictable. Do not use this field as output when the converter was called from a CICS business logic interface that was called in offset mode.

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, this length 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, do not use this pointer.

decode_input_data_len

(Input and output)

On input, this is the length in bytes of the request data pointed to by **decode_data_ptr**.

The value to be used for the **DATALENGTH** option of the **EXEC CICS LINK** command for the CICS program. The default value if this output is not set is 32KB.

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, this length 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, do not use this pointer.

decode_output_data_len

(Output only)

The value to be used for the **LENGTH** option of the **EXEC CICS LINK** command for the CICS program. The default value if this output is not set is 32KB.

decode_reason

(Output only)

A reason code—see "Responses and reason codes" on page 125.

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, this length is zero.

decode_request_header_ptr

(Input only)

A pointer to the first HTTP header in the HTTP request. If the analyzer modified this part of the request, the changes are visible here. If **decode_request_header_length** is zero, do not use this pointer.

decode_resource_length

(Input only)

The length in bytes of the string identifying the HTTP absolute path specified in the HTTP request. If the request is not an HTTP request, this length is zero.

decode_resource_ptr

(Input only)

A pointer to the string identifying the HTTP absolute path specified in the HTTP request. If the analyzer modified this part of the request, the changes are visible here. If **decode_resource_length** is zero, do not use this pointer.

decode_response

(Output only)

A response—see “Responses and reason codes” on page 125.

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 CICS program that is to service the request. The CICS program name must be set here or in the analyzer, or no CICS program will be called.

decode_user_data_length

(Input only)

The length in bytes of the user data for this HTTP request. If the analyzer modified this value, it is visible here. If there is no user data in the request, the length is zero. If the request is not an HTTP request, this length is the length of the request.

decode_user_data_ptr

(Input only)

A pointer to any user data for this HTTP request. If the analyzer modified this part of the request, the changes are visible here. If there is no user data in the request, the pointer is zero. If the request is not an HTTP request, this pointer has the same value as **decode_data_ptr**.

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 single-character parameter list version identifier, which changes whenever the layout of the parameter list changes. Its value is a character zero (X'F0'), indicating a CICS TS 1.1.1 version parameter list.

decode_volatile

(Input)

A single-character code indicating whether the data area pointed to by **decode_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 **decode_data_ptr** can be freed and replaced by a different size workarea.

Responses and reason codes

You must return one of the following values in **decode_response**:

URP_OK

The alias, or the CICS business logic interface, links to the CICS program using the communication area provided by **Decode**.

URP_EXCEPTION

The CICS program is not executed.

If the alias was the caller, the action taken depends on the reason code:

- **URP_SECURITY_FAILURE**—the alias writes an exception trace entry (trace point 455A), and issues a message (DFHWB0121). If the request is an HTTP request, response 403 is sent to the Web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.
- **URP_CORRUPT_CLIENT_DATA**—the alias writes an exception trace entry (trace point 4559), and issues a message (DFHWB0121). If the request is an HTTP request, response 400 is sent to the Web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.
- Any other value—the alias writes an exception trace entry (trace point 455B), and issues a message (DFHWB0121). If the request is an HTTP request, response 501 is sent to the Web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If the CICS business logic interface was the caller, the action taken depends on the reason code:

- **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—the CICS business logic interface writes an exception trace entry (trace point 455B), issues a message (DFHWB0121), and returns a response of 501 to its caller.

URP_INVALID

The CICS program is not executed.

If the alias was the caller, the alias writes an exception trace entry (trace point 455C), and issues a message (DFHWB0121). If the request is an

HTTP request, response 501 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If the CICS business logic interface was the caller, the CICS business logic interface writes an exception trace entry (trace point 455C), issues a message (DFHWB0121), and returns a response of 501 to its caller.

URP_DISASTER

The CICS program is not executed.

If the alias was the caller, the alias writes an exception trace entry (trace point 455D), and issues a message (DFHWB0121). If the request is an HTTP request, response 501 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If the CICS business logic interface was the caller, the CICS business logic interface writes an exception trace entry (trace point 455D), issues a message (DFHWB0121), and returns a response of 501 to its caller.

If you return any other value in **decode_response**, the CICS program is not executed.

If the alias was the caller, the alias writes an exception trace entry (trace point 455E), and issues a message (DFHWB0121). If the request is an HTTP request, response 500 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If the CICS business logic interface was the caller, the CICS business logic interface writes an exception trace entry (trace point 455E), issues a message (DFHWB0121), and returns a response of 501 to its caller.

You may supply a 32-bit reason code in **decode_reason** to provide further information in error cases. Neither the 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 output in any trace entry that results from the invocation of **Decode**.

See “Numeric values of symbolic codes” on page 88 for the numeric values of the response and reason codes in trace output.

Encode

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 5. Parameters for **Encode**

Input encode_	Inout encode_	Output encode_
eyecatcher entry_countfunction input_data_len user_token	data_ptr	reason response

Function

If the analyzer, or the caller of the CICS business logic interface, specified a converter name for the request, **Encode** is called after the CICS program has ended. It constructs the response from the contents of the communication area.

Parameters

encode_data_ptr

(Input and output)

On input, a pointer to the communication area returned by the CICS program. If no CICS program was called, it is a pointer to the communication area created by **Decode**.

On output, a pointer to the buffer containing the response to be sent to the client. 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. 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** converter 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 communication area as specified by **Decode** in **decode_output_data_len**.

encode_reason

(Output only)

A reason code—see “Responses and reason codes”.

encode_response

(Output only)

A response—see “Responses and reason codes”.

encode_user_token

(Input only)

The 64-bit token output by **Decode** 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 is a character zero (X'F0'), indicating a CICS TS 1.1.1 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 workarea.

Responses and reason codes

You must return one of the following values in **encode_response**:

URP_OK

The response in the buffer pointed to by **encode_data_ptr** is sent to the client.

URP_DISASTER

If the alias was the caller, the alias writes an exception trace entry (trace point 455D), and issues a message (DFHWB0122). If the request is an HTTP request, response 501 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If the CICS business logic interface was the caller, the CICS business logic interface writes an exception trace entry (trace point 455D), issues a message (DFHWB0122), and returns a response of 501 to its caller.

URP_OK_LOOP

The CICS Web interface 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.

If the alias was the caller and you return any other value in **encode_response**, the alias writes an exception trace entry (trace point 455E), and issues a message (DFHWB0122). If the request is an HTTP request, response 501 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the TCP/IP for VSE/ESA socket is closed.

If the CICS business logic interface was the caller and you return any other value in **encode_response**, the CICS business logic interface 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 the CICS Web support nor the CICS business logic interface takes any action on the reason code returned by **Encode**. The reason code is output in any trace entry that results from the invocation of **Encode**.

See “Numeric values of symbolic codes” on page 88 for the numeric values of the response and reason codes in trace output.

Appendix D. Reference information for DFHWBTL

The HTML template manager helps you to write CICS application programs that create HTML pages to be sent to an HTTP client. You use EXEC CICS LINK to call DFHWBTL.

An HTML page can be built from one or more templates. The templates can be read from a VSE/ESA sub-library, or can be provided inline in your application program, or can be defined in a DOCTEMPLATE definition. DOCTEMPLATEs define templates with 48-character names. The template name used in DFHWBTL is padded with 40 blanks and the corresponding DOCTEMPLATE is used if it exists. If there is no corresponding DOCTEMPLATE, a resource definition for the member in the DFHHTML library is created dynamically.

Templates can contain HTML symbols, and the template manager replaces the symbols with values from a symbol table as it adds the template to a page. The template manager allows you to set up and modify a symbol table as you add templates to the HTML page.

The functions of the template manager are summarized as follows:

- BUILD_HTML_PAGE combines the functions of START_HTML_PAGE, ADD_HTML_TEMPLATE, and END_HTML_PAGE.
- START_HTML_PAGE establishes an environment for the next three functions, and allows you to put values in the symbol table.
- ADD_HTML_SYMBOLS adds symbols to the symbol table. It also modifies the values of symbols already defined in the symbol table.
- ADD_HTML_TEMPLATE adds a template to the HTML page, replacing symbols in the template with the values defined in the symbol table.
- END_HTML_PAGE destroys the environment established in START_HTML_PAGE, though the page remains in the storage in which it was constructed.

You call the template manager using EXEC CICS LINK as follows:

```
EXEC CICS LINK PROGRAM(DFHWBTL) COMMAREA(...) LENGTH(...)
```

You supply the communication area addressed by the COMMAREA option of the command. The contents of the communication area are described below.

In this chapter the various program elements (values) are given symbolic names. These names, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of the CICS Web support. The files for the various languages are as follows:

Language	File
Assembler	DFHWBTLD
C	DFHWBTLH
COBOL	DFHWBTLO
PL/I	DFHWBTLL

These files give language-specific information about the data types of the fields in the communication area.

Parameters in the communication area

The following table summarizes the use of the parameters by function.

Table 6. Parameters for the HTML template manager

Function	Parameters	Usage
WBTL_START_HTML_PAGE	wbtl_version_no wbtl_function wbtl_response wbtl_reason wbtl_connect_token wbtl_symbol_list_ptr wbtl_symbol_list_len	input input output output output input input
WBTL_ADD_HTML_SYMBOLS	wbtl_version_no wbtl_function wbtl_response wbtl_reason wbtl_connect_token wbtl_symbol_list_ptr wbtl_symbol_list_len	input input output output input input input
WBTL_ADD_HTML_TEMPLATE	wbtl_version_no wbtl_function wbtl_response wbtl_reason wbtl_connect_token wbtl_template_name wbtl_template_abstime wbtl_template_buffer_ptr wbtl_template_buffer_len wbtl_html_buffer_ptr wbtl_html_buffer_len	input input output output input input input input inout inout
WBTL_END_HTML_PAGE	wbtl_version_no wbtl_function wbtl_response wbtl_reason wbtl_connect_token	input input output output input
WBTL_BUILD_HTML_PAGE	wbtl_version_no wbtl_function wbtl_response wbtl_reason wbtl_template_name wbtl_template_abstime wbtl_template_buffer_ptr wbtl_template_buffer_len wbtl_symbol_list_ptr wbtl_symbol_list_len wbtl_html_buffer_ptr wbtl_html_buffer_len	input input output output input output input input input input inout inout

wbtl_version_no
(Input only)

The version number of the template manager interface. Specify WBTL_CURRENT_VERSION.

wbtl_function
(Input only)

Specify the function you wish to perform as one of the following:

- `WBTL_BUILD_HTML_PAGE`
- `WBTL_START_HTML_PAGE`
- `WBTL_ADD_HTML_SYMBOLS`
- `WBTL_ADD_HTML_TEMPLATE`
- `WBTL_END_HTML_PAGE`

See “Numeric values of symbolic codes” on page 88 for the numeric values of the functions in trace output.

wbtl_response

(Output only)

The response from the template manager to the function and inputs. See “Responses and reason codes” on page 134.

wbtl_reason

(Output only)

Might contain additional information about an error for some responses. See “Responses and reason codes” on page 134.

wbtl_connect_token

(Input and output)

As output from `WBTL_START_HTML_PAGE`, this token represents the page environment established by `WBTL_START_HTML_PAGE`, and you must save it for use with other functions. You can have several tokens in use at once, and the template manager maintains separate page environments for each token.

As input to `WBTL_ADD_HTML_SYMBOLS`, `WBTL_ADD_HTML_TEMPLATE`, and `WBTL_END_HTML_PAGE`, this token identifies the HTML page environment.

wbtl_template_name

(Input only)

As optional input to `WBTL_BUILD_HTML_PAGE`, and `WBTL_ADD_HTML_TEMPLATE`, this is an 8-character field, padded on the right with spaces. If you want the template manager to use a template from a VSE/ESA sub-library, put the name of the member here. If you want the template manager to use an inline template, put spaces here and use the **wbtl_template_buffer_ptr** and **wbtl_template_buffer_len** fields.

wbtl_template_abstime

(Output only)

As output from `WBTL_ADD_HTML_TEMPLATE` and `WBTL_BUILD_HTML_PAGE` when the template manager is requested to use the VSE/ESA sub-library member specified by `wbtl_template_name`. This is the current date and time.

wbtl_template_buffer_ptr

(Input only)

As optional input to `WBTL_BUILD_HTML_PAGE` and `WBTL_ADD_HTML_TEMPLATE`, this is the address of the template to be used. If you want the template manager to use an inline template, use this field. If you want the template manager to use a template from a VSE/ESA sub-library, do not use this field, but use **wbtl_template_name** instead. This field is ignored if **wbtl_template_name** is specified.

wbtl_template_buffer_len

(Input only)

As optional input to WBTL_BUILD_HTML_PAGE and WBTL_ADD_HTML_TEMPLATE, this is the length in bytes of the template pointed to by **wbtl_template_buffer_ptr**. If you want the template manager to use an inline template, use this field. If you want the template manager to use a template from a VSE/ESA sub-library, do not use this field, but use **wbtl_template_name** instead. This field is ignored if **wbtl_template_name** is specified.

wbtl_symbol_list_ptr

(Input only)

This field is a required input to WBTL_ADD_HTML_SYMBOLS, and an optional input to WBTL_BUILD_HTML_PAGE and WBTL_START_HTML_PAGE. It is the address of the list of symbols to be used to update the symbol table. The format of the list is described in "Describing symbols, symbol table, and symbol list" on page 74. If the function is WBTL_ADD_HTML_SYMBOLS, you must use **wbtl_connect_token** to identify the page environment whose symbol table is to be updated.

wbtl_symbol_list_len

(Input only)

This field is a required input to WBTL_ADD_HTML_SYMBOLS, and an optional input to WBTL_BUILD_HTML_PAGE and WBTL_START_HTML_PAGE. It is the length in bytes of the list of symbols to be used to update the symbol table.

wbtl_html_buffer_ptr

(Input and output)

As input to WBTL_BUILD_HTML_PAGE and WBTL_ADD_HTML_TEMPLATE, this field is the address of the unused portion of the buffer that contains the HTML page being constructed. As output from WBTL_BUILD_HTML_PAGE and WBTL_ADD_TEMPLATE, this field is the address of the remaining space in the buffer.

wbtl_html_buffer_len

(Input and output)

As input to WBTL_BUILD_HTML_PAGE and WBTL_ADD_HTML_TEMPLATE, this is the length in bytes of the unused portion of the buffer that contains the HTML page being constructed. As output from WBTL_BUILD_HTML_PAGE and WBTL_ADD_HTML_TEMPLATE, this is the length in bytes of the remaining space in the buffer.

Responses and reason codes

WBTL_OK

The operation ended successfully.

WBTL_EXCEPTION

The template manager detected an error in the operation. The following reason values are possible:

WBTL_PAGE_TRUNCATED

There was not enough room left in the buffer for the page. The HTML template manager has used all the space available, and discarded the rest of the page.

WBTL_TEMPLATE_NOT_FOUND

The template manager could not find the template named in **wbtl_template_name** in the VSE/ESA sub-library.

WBTL_TEMPLATE_TRUNCATED

There was not enough room left in the buffer for the template. The HTML template manager has used all the space available, and discarded the rest of the template.

WBTL_INVALID

The template manager detected an error in the parameters in the communication area. The following reason values are possible:

WBTL_INVALID_BUFFER_PTR

The value in **wbtl_html_buffer_ptr** was zero when an address was required.

WBTL_INVALID_FUNCTION

The value in **wbtl_function** was not recognized.

WBTL_INVALID_SYMBOL_LIST

An input symbol list was required, but either **wbtl_symbol_list_ptr** was zero, or **wbtl_symbol_list_len** was zero.

WBTL_INVALID_TOKEN

The operation was expecting an input **wbtl_connect_token**, but found its value was zero. All tokens output by the HTML template manager are non-zero.

WBTL_DISASTER

The template manager detected an unrecoverable error. The following reason values are possible:

WBTL_FREEMAIN_ERROR

There was an error while attempting to release storage.

WBTL_GETMAIN_ERROR

There was an error while attempting to acquire storage.

See “Numeric values of symbolic codes” on page 88 for the numeric values of the response and reason codes in trace output.

Appendix E. Reference information for DFHWBENV

The environment variables program is DFHWBENV. It extracts information about the server (the CICS region in which the server controller is running), and the client (the Web browser that sent the current request). You can use EXEC CICS LINK to call it. You must supply a communication area that is long enough to contain the expected response. The exact length of the response depends on the nature of your connection with the client, and the values set by the client's browser program, but 1024 bytes will usually be enough. On return, the communication area contains a 32-bit integer followed by a sequence of values of environment variables. The 32-bit integer specifies the length of the string that follows it. The values are specified with the following format:

variable-name=value

Each value is separated from the following variable name by an ampersand. None of the values contain an ampersand. This format is the same as that required for input as a symbol list to the HTML template manager (DFHWBTL), and to the parser (DFHWBPA). If the environment variables program cannot return any variables, it returns a length of zero. If the communication area you provide is not long enough to contain all the variables and their values, the program abends with abend code AWBC.

DFHWBENV can be linked to only from the alias transaction. You cannot link to DFHWBENV from the analyzer.

The meaning of the value for each variable name provided by CICS that can occur in the communication area is as follows:

CONTENT_LANGUAGE

The national language of any user data in the HTTP request. The value contains the ISO 3316 language code, optionally qualified by an ISO 639 country code. It is extracted from the Content-Language HTTP header. If there is no Content-Language header, the value is a null string.

CONTENT_LENGTH

The character representation of the decimal length of any user data in the HTTP request. It is extracted from the Content-Length HTTP header. If there is no user data, the value is zero.

CONTENT_TYPE

The MIME format of any user data in the HTTP request. It is extracted from the Content-Type HTTP header. If there is no user data, the value is a null string.

QUERY_STRING

The query string from the HTTP request. Any ampersands in the query string are expanded to %26;, and any equals signs are expanded to %3D;. If there is no query string, the value is a null string.

REMOTE_ADDR

The IP address of the client in dotted decimal format.

REMOTE_HOST

The fully-qualified name of the client, if this can be obtained from the name server. If the name cannot be found, the value is a null string.

REMOTE_USER

The user ID that has been assigned to the current request.

REQUEST_METHOD

The method name specified in the first HTTP header received from the client. It is one of GET, POST, HEAD, SHOWMETHOD, PUT, DELETE, LINK, UNLINK.

SERVER_NAME

The fully-qualified name of the connection, for example `www.hursley.ibm.com`. If CICS was unable to obtain its own name from the domain name server when the CICS Web support was enabled, the dotted decimal address of the connection will be returned instead.

SERVER_PORT

The character representation of the decimal value of the TCP/IP port on which the request was received, for example 80.

SERVER_PROTOCOL

The name of the Internet protocol describing the data received, usually HTTP/1.0.

SERVER_SOFTWARE

The name and version of the CICS product.

All HTTP headers found in the inbound request are also placed in the commarea, and are given the prefix **HTTP_**. Any variables passed in an HTTP request that do not conform to RFC 1945 naming standards are ignored by DFHWBENV and are not returned in the commarea. Some examples of valid headers are:

HTTP_ACCEPT

The contents of all the Accept HTTP headers, separated by commas. These values represent the MIME types that the browser is prepared to accept, so the list should never be empty. However, if there are no Accept headers, the value is a null string.

HTTP_ACCEPT_ENCODING

The contents of the Accept-Encoding HTTP header. If there is no Accept-Encoding header, the variable is not returned.

HTTP_ACCEPT_LANGUAGE

The contents of the Accept-Language HTTP header. If there is no Accept-Language header, the variable is not returned.

HTTP_AUTHORIZATION

The contents of the Authorization HTTP header. If there is no Authorization header, the variable is not returned.

HTTP_CHARGE_TO

The contents of the Charge-To HTTP header. If there is no Charge-To header, the variable is not returned.

HTTP_FROM

The contents of the From HTTP header. If there is no From header, the variable is not returned.

HTTP_IF_MODIFIED_SINCE

The contents of the If-Modified-Since HTTP header. If there is no If-Modified-Since header, the variable is not returned.

HTTP_PRAGMA

The contents of the Pragma HTTP header. If there is no Pragma header, the variable is not returned.

HTTP_REFERER

The contents of the Referer HTTP header. This is the URL of the page from which the link was made. If there is no Referer header, the variable is not returned.

HTTP_USER_AGENT

The contents of the User-Agent HTTP header. This is the product name of the Web browser program. If there is no User-Agent header, the variable is not returned.

Appendix F. Reference information for DFH\$WBST and DFH\$WBSR

Two state management sample programs, DFH\$WBST and DFH\$WBSR are supplied with the CICS Web Interface. 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 EXEC CICS GETMAIN 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 program and the tokens it allocates can be used in many ways. Here are two suggestions:

- The token can be used as a *conversation token*, that is a token that identifies information that is to be preserved throughout a pseudoconversation. A conversation token can be managed by the converter or the CICS program, and is best conveyed from program to program in a pseudoconversation as a hidden field in an HTML form.
- The token can be used as a *session token*, that is a token that identifies information that is to be preserved throughout an extended interaction between an end user and various CICS programs, perhaps over several pseudoconversations. A session token can be managed by the analyzer, and is best conveyed from interaction to interaction as a query string in a URL. This use of a state management token is illustrated by the security analyzer, security converter, and security sign-on sample programs described in “Using sample programs for security” on page 82.

The state management program provides 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.
- Remove information and tokens that have expired.

The last operation is an internal operation, not explicitly invoked by the caller.

The layout of the 268-byte communication area is shown in the following table. You must clear the communication area to binary zeros before setting the inputs for the function you require.

Table 7. 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.

Table 7. Parameters for the state management program (continued)

Offset	Length	Type	Value	Notes
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 a Create call, 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 user data associated with the input token at offset 8 is now in the user data area at offset 12.
 - If the function was Store, the input user data at offset 12 is now associated with the input token and offset 8. Any user data 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 communication area.
- 3** The function was Create, but EXEC CICS GETMAIN 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** EXEC CICS WRITEQ TS gave an error response when writing internal data to a temporary storage queue.
- 7** EXEC CICS ASKTIME gave an error response.
- 8** EXEC CICS READQ TS gave an error response when reading internal data from a temporary storage queue.
- 9** EXEC CICS ASKTIME gave an error response during timeout processing.
- 11** The function was Create, but EXEC CICS WRITEQ TS gave an error response. This return code is produced only by DFH\$WBSR.
- 12** The function was Retrieve, but EXEC CICS READQ TS gave an error response. This return code is produced only by DFH\$WBSR.
- 13** The function was Store, but EXEC CICS WRITEQ TS gave an error response. This return code is produced only by DFH\$WBSR.
- 14** The function was Destroy, but EXEC CICS DELETEQ TS gave an error response. This return code is produced only by DFH\$WBSR.

Appendix G. Reference information for DFHWBPA

The CICS Web support parser program is DFHWBPA. It parses strings of the form:
key1=value1&key2=value2&key3=value3 ...

key1 is a keyword, value1 is the corresponding value, and so on. The keyword/value pairs must be separated by ampersands as shown in the example. If there is only one keyword/value pair there must be no ampersand. A keyword must contain only uppercase and lowercase letters, digits, and underscores (“_”). It must not contain any imbedded blanks. A value can contain any character except an ampersand. The kinds of strings that the parser accepts are the same as:

- Data transmitted by HTTP clients as query strings
- Forms data from HTTP clients
- Output from the environment variables program DFHWBENV
- Input to the HTML template manager

The parser accepts a string and a keyword as input, and returns the corresponding value as output. If the string does not contain the keyword, the output is nulls.

The program is called by EXEC CICS LINK. You supply a communication area containing the keyword to be found, two ampersands, and the string to be searched. The communication area must not be more than 4096 bytes long.

```
EXEC CICS LINK PROGRAM(DFHWBPA) COMMAREA(...) LENGTH(...)
```

When the parser returns to your program, the communication area contains the value followed by nulls.

The following example illustrates the operation of the parser. Suppose the input communication area contains the following string:

```
a1&myt=New Authors&a1=Halliwell Sutcliffe&a2=Stanley Weyman
```

The output is:

```
Halliwell Sutcliffe
```

The output is padded to 60 bytes (the length of the input communication area) with nulls.

Appendix H. Reference information for DFHWBEP

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.

The names of the parameters and constants in the communication area passed to 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**, which is input and output.

wbep_abend_code

(Input only)

The 8-character abend code associated with this exception.

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)

The 15-character TCPIP address of the client.

wbep_client_address_len

(Input only)

The length of the TCP/IP address contained in WBEP_CLIENT_ADDRESS.

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, if invoked.

wbep_converter_response

(Input only)

The response code returned by the converter, 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)

The program in which the exception occurred.

wbep_http_response_code

(Input only)

The HTTP error response code returned by CICS for this error. You can change this response code by manipulating the response in the buffer pointed to by WBEP_RESPONSE_PTR.

wbep_length

(Input only)

The length of the DFHWBEPC copybook.

wbep_message_len

(Input only)

The length of the message addressed by WBEP_MESSAGE_PTR.

wbep_message_number

(Input only)

A fullword number of the CICS WB message associated with the error.

wbep_message_ptr

(Input only)

A pointer to the CICS message text associated with this exception.

wbep_response_len

(Input only)

The fullword length of the CICS message text associated with this exception.

wbep_response_ptr

(Input and output)

A pointer to the response message text associated with this exception.

wbep_server_address

(Input only)

The 15-character TCPIP address of the server.

wbep_server_address_len

(Input only)

The length of the TCP/IP address contained in WBEP_SERVER_ADDRESS.

wbep_target_program

(Input only)

The target program associated with the Web request.

wbep_tcpipservice_name

(Input only)

The name of the TCPIPSERVICE associated with this request.

wbep_version

(Input only)

The version of DFHWBEP being passed by CICS.

Appendix I. HTML-coded character sets

Table 8 lists the supported IANA charset= values and the IBM CCSID equivalents. All of these values are valid for codepage conversions on the following commands:

- EXEC CICS WEB SEND
- EXEC CICS WEB RECEIVE
- EXEC CICS DOCUMENT RETRIEVE

On the CLNTCODEPAGE parameter of these commands, you can specify either the IANA value or the IBM CCSID value, as CICS performs mapping between the two.

Table 8. 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	x-sjis or shift-jis	943 (932, a subset of 943, is also valid)
	EUC Japanese	euc-jp	5050 (EUC)
Korean	EUC Korean	euc-kr	970 (for AIX or Unix)
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

Table 8. Coded character sets (continued)

Language	Coded character set	IANA charset	IBM CCSID
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
Swedish	ISO/IEC 8859-1	iso-8859-1	819
Turkish	ISO/IEC 8859-9	iso-8859-9	920
Ukrainian	Windows 1251	windows-1251	1251
	UCS-2	iso-10646-ucs-2	1200 (growing) or 13488 (fixed)

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 to any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact Laboratory Counsel, MP151, IBM United Kingdom Limited, 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.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

Programming interface information

This book is intended to help you use the external interfaces provided by the CICS Transaction Server for VSE/ESA. This book documents General-use Programming Interface and Associated Guidance Information provided by CICS.

General-use programming interfaces allow the customer to write programs that obtain the services of CICS.

This book also documents Product-sensitive Programming Interface and Associated Guidance Information and Diagnosis, Modification or Tuning Information provided by CICS.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of CICS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified, where it occurs, by an introductory statement to a chapter or section.

Diagnosis, Modification, or Tuning Information is provided to help you diagnose problems in your CICS system.

Note: Do not use this Diagnosis, Modification, or Tuning Information as a programming interface.

Diagnosis, Modification, or Tuning Information is identified, where it occurs, by an introductory statement to a chapter or section.

Trademarks and service marks

The following terms, used in this publication, are trademarks or service marks of IBM Corporation in the United States or other countries:

BookManager	CICS
CICS/VSE	DB2
DFS	IBM
IMS	Language Environment
OS/2	RACF
RT	SAA
VTAM	

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Bibliography

CICS Transaction Server for VSE/ESA Release 1 Library

Evaluation and planning	
<i>CICS Release Guide</i>	GC33-1645
<i>CICS Migration Guide</i>	GC33-1646
<i>CICS Report Controller Planning Guide</i>	SC33-1941
<i>CICS Enhancements Guide</i>	GC34-5763
General	
<i>CICS Master Index</i>	SC33-1648
<i>CICS Trace Entries Handbook</i>	SC34-5556
<i>CICS User's Handbook</i>	SX33-6101
<i>CICS Glossary</i> (softcopy only)	GC33-1649
Administration	
<i>CICS System Definition Guide</i>	SC33-1651
<i>CICS Customization Guide</i>	SC33-1652
<i>CICS Resource Definition Guide</i>	SC33-1653
<i>CICS Operations and Utilities Guide</i>	SC33-1654
<i>CICS Supplied Transactions</i>	SC33-1655
Programming	
<i>CICS Application Programming Guide</i>	SC33-1657
<i>CICS Application Programming Reference</i>	SC33-1658
<i>CICS 4.1 Sample Applications Guide</i>	SC33-1713
<i>CICS Application Migration Aid Guide</i>	SC33-1943
<i>CICS System Programming Reference</i>	SC33-1659
<i>CICS Distributed Transaction Programming Guide</i>	SC33-1661
<i>CICS Front End Programming Interface User's Guide</i>	SC33-1662
<i>REXX Guide</i>	SC34-5764
<i>CICS Family: Client/Server Programming</i>	SC33-1435
Diagnosis	
<i>CICS Problem Determination Guide</i>	GC33-1663
<i>CICS Messages and Codes</i>	SC33-6799
<i>CICS Diagnosis Reference</i>	LY33-6085
<i>CICS Data Areas</i>	LY33-6086
<i>CICS Supplementary Data Areas</i>	LY33-6087
Communication	
<i>CICS Internet Guide</i>	SC34-5765
<i>CICS Intercommunication Guide</i>	SC33-1665
<i>CICS Family: Interproduct Communication</i>	SC33-0824
<i>CICS Family: Communicating from CICS on System/390</i>	SC33-1697
Special topics	
<i>CICS Recovery and Restart Guide</i>	SC33-1666
<i>CICS Performance Guide</i>	SC33-1667
<i>CICS Shared Data Tables Guide</i>	GC33-1668
<i>CICS Security Guide</i>	SC33-1942
<i>CICS External Interfaces Guide</i>	SC33-1669
<i>CICS XRF Guide</i>	SC33-1671
<i>CICS Report Controller User's Guide</i>	SC34-5688
CICS Clients	
<i>CICS Clients: Administration</i>	SC33-1792
<i>CICS Universal Clients Version 3 for OS/2: Administration</i>	SC34-5450
<i>CICS Universal Clients Version 3 for Windows:</i>	SC34-5449
Administration	

<i>CICS Universal Clients Version 3 for AIX: Administration</i>	SC34-5348
<i>CICS Universal Clients Version 3 for Solaris: Administration</i>	SC34-5451
<i>CICS Family: OO programming in C++ for CICS Clients</i>	SC33-1923
<i>CICS Family: OO programming in BASIC for CICS Clients</i>	SC33-1671
<i>CICS Family: Client/Server Programming</i>	SC33-1435
<i>CICS Transaction Gateway Version 3: Administration</i>	SC34-5448

Books from VSE/ESA 2.5 base program libraries

VSE/ESA Version 2 Release 5

Book title	Order number
Administration	SC33-6705
Diagnosis Tools	SC33-6614
Extended Addressability	SC33-6621
Guide for Solving Problems	SC33-6710
Guide to System Functions	SC33-6711
Installation	SC33-6704
Licensed Program Specification	GC33-6700
Messages and Codes Volume 1	SC33-6796
Messages and Codes Volume 2	SC33-6798
Messages and Codes Volume 3	SC33-6799
Networking Support	SC33-6708
Operation	SC33-6706
Planning	SC33-6703
Programming and Workstation Guide	SC33-6709
System Control Statements	SC33-6713
System Macro Reference	SC33-6716
System Macro User's Guide	SC33-6715
System Upgrade and Service	SC33-6702
System Utilities	SC33-6717
TCP/IP User's Guide	SC33-6601
Turbo Dispatcher Guide and Reference	SC33-6797
Unattended Node Support	SC33-6712
e-business Connectors User's Guide	SC33-6719

High-Level Assembler Language (HLASM)

Book title	Order number
General Information	GC26-8261
Installation and Customization Guide	SC26-8263
Language Reference	SC26-8265
Programmer's Guide	SC26-8264

Language Environment for VSE/ESA (LE/VSE)

Book title	Order number
C Run-Time Library Reference	SC33-6689
C Run-Time Programming Guide	SC33-6688
Concepts Guide	GC33-6680
Debug Tool for VSE/ESA Fact Sheet	GC26-8925
Debug Tool for VSE/ESA Installation and Customization Guide	SC26-8798
Debug Tool for VSE/ESA User's Guide and Reference	SC26-8797
Debugging Guide and Run-Time Messages	SC33-6681
Diagnosis Guide	SC26-8060
Fact Sheet	GC33-6679
Installation and Customization Guide	SC33-6682
LE/VSE Enhancements	SC33-6778
Licensed Program Specification	GC33-6683
Programming Guide	SC33-6684
Programming Reference	SC33-6685
Run-Time Migration Guide	SC33-6687
Writing Interlanguage Communication Applications	SC33-6686

VSE/ICCF

Book title	Order number
Administration and Operations	SC33-6738
User's Guide	SC33-6739

VSE/POWER

Book title	Order number
Administration and Operation	SC33-6733
Application Programming	SC33-6736
Networking Guide	SC33-6735
Remote Job Entry User's Guide	SC33-6734

VSE/VSAM

Book title	Order number
Commands	SC33-6731
User's Guide and Application Programming	SC33-6732

VTAM for VSE/ESA

Book title	Order number
Customization	LY43-0063
Diagnosis	LY43-0065
Data Areas	LY43-0104
Messages and Codes	SC31-6493
Migration Guide	GC31-8072
Network Implementation Guide	SC31-6494
Operation	SC31-6495
Overview	GC31-8114
Programming	SC31-6496
Programming for LU6.2	SC31-6497
Release Guide	GC31-8090
Resource Definition Reference	SC31-6498

Books from VSE/ESA 2.5 optional program libraries

C for VSE/ESA (C/VSE)

Book title	Order number
C Run-Time Library Reference	SC33-6689
C Run-Time Programming Guide	SC33-6688
Diagnosis Guide	GC09-2426
Installation and Customization Guide	GC09-2422
Language Reference	SC09-2425
Licensed Program Specification	GC09-2421
Migration Guide	SC09-2423
User's Guide	SC09-2424

COBOL for VSE/ESA (COBOL/VSE)

Book title	Order number
Debug Tool for VSE/ESA Fact Sheet	GC26-8925
Debug Tool for VSE/ESA Installation and Customization Guide	SC26-8798
Debug Tool for VSE/ESA User's Guide and Reference	SC26-8797
Diagnosis Guide	SC26-8528
General Information	GC26-8068
Installation and Customization Guide	SC26-8071
Language Reference	SC26-8073
Licensed Program Specifications	GC26-8069
Migration Guide	GC26-8070
Migrating VSE Applications To Advanced COBOL	GC26-8349
Programming Guide	SC26-8072

DB2 Server for VSE

Book title	Order number
Application Programming	SC09-2393
Database Administration	GC09-2389
Installation	GC09-2391
Interactive SQL Guide and Reference	SC09-2410
Operation	SC09-2401
Overview	GC08-2386
System Administration	GC09-2406

DL/I VSE

Book title	Order number
Application and Database Design	SH24-5022
Application Programming: CALL and RQDLI Interface	SH12-5411
Application Programming: High-Level Programming Interface	SH24-5009
Database Administration	SH24-5011
Diagnostic Guide	SH24-5002
General Information	GH20-1246
Guide for New Users	SH24-5001
Interactive Resource Definition and Utilities	SH24-5029
Library Guide and Master Index	GH24-5008
Licensed Program Specifications	GH24-5031
Low-level Code and Continuity Check Feature	SH20-9046
Library Guide and Master Index	GH24-5008
Messages and Codes	SH12-5414
Recovery and Restart Guide	SH24-5030
Reference Summary: CALL Program Interface	SX24-5103
Reference Summary: System Programming	SX24-5104
Reference Summary: HLPI Interface	SX24-5120
Release Guide	SC33-6211

PL/I for VSE/ESA (PL/I VSE)

Book title	Order number
Compile Time Messages and Codes	SC26-8059
Debug Tool For VSE/ESA User's Guide and Reference	SC26-8797
Diagnosis Guide	SC26-8058
Installation and Customization Guide	SC26-8057
Language Reference	SC26-8054

Book title	Order number
Licensed Program Specifications	GC26-8055
Migration Guide	SC26-8056
Programming Guide	SC26-8053
Reference Summary	SX26-3836

Screen Definition Facility II (SDF II)

Book title	Order number
VSE Administrator's Guide	SH12-6311
VSE General Introduction	SH12-6315
VSE Primer for CICS/BMS Programs	SH12-6313
VSE Run-Time Services	SH12-6312

TCP/IP for VSE/ESA

Documentation for TCP/IP for VSE/ESA is available by ordering the "TCP/IP for VSE/ESA PDF Product Library" product kit, SK2T-1336. **Not** all the books are separately orderable as they are only available in PDF form. The following titles are available on the product kit at the time of going to press:

Book title	Order number
TCP/IP for VSE/ESA — IBM Program Setup and Supplementary Information	SC33-6601
TCP/IP for VSE/ESA Flyer	G221-9030
TCP/IP for VSE/ESA Performance Considerations	n/a
TCP/IP for VSE/ESA LPS	GC33-6594
The Native TCP/IP forVSE	GC33-6594
TCP/IP Tutorial and Technical Overview	GG24-3376
VSE/ESA as a Web Server	SG24-2040
TCP/IP for VSE 1.4 Installation Guide	n/a
TCP/IP for VSE 1.4 User's Guide	n/a
TCP/IP for VSE 1.4 Commands	n/a
TCP/IP for VSE 1.4 Programmer's Reference	n/a
TCP/IP for VSE 1.4 Messages and Codes	n/a
TCP/IP for VSE 1.4 Optional Products	n/a

Information on the World Wide Web

Information about the hypertext transfer protocol (HTTP) and the hypertext markup language (HTML) is on the World Wide Web. URLs are provided in this book with the caveat that their permanence cannot be guaranteed.

HTTP/1.0

CICS supports HTTP/1.0. Unpredictable results can occur if you use HTTP/1.1-specific headers. For HTTP/1.0 information, consult the following:

- *Overview of HTTP*
<http://www.w3.org/hypertext/WWW/Protocols/Overview.html>
- *Hypertext Transfer Protocol (HTTP/1.0)*
<http://ds.internic.net/rfc/rfc1945.txt>

The following references are to information about the ISO 8859-1 (Latin-1) character set:

- *ISO 8859-1 National Character Set FAQ*
<http://aliga.cesca.es:1025/%7Ezopcgp01/manuals/ISO8859-1.faq>
- *ISO 8859-1:1987 (ordering information)*
<http://www.iso.ch/cate/d16338.html>
- *ISO 8859-1 (Latin-1) Characters List*
http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/iso_table.html
- *Table of Latin-1 character glyphs*
<http://www.w3.org/pub/WWW/MarkUp/Wilbur/latin1.gif>

HTML

CICS has no dependency on the level of HTML used. There are many reference sites available on the World Wide Web. The most useful starting place for HTML reference material is:

<http://www.w3.org/>

Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager softcopy versions of a publication are in step, but subsequent updates will probably be available in softcopy before they are available in hardcopy.

For CICS for VSE/ESA books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx and on the

VSE/ESA Collection Kit CD-ROM, SK2T-0060-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Here's how to determine if you are looking at the most current copy of a publication:

- A publication with a higher suffix number is more recent than one with a lower suffix number. For example, the publication with order number SC33-0667-02 is more recent than the publication with order number SC33-0667-01. (Note that suffix numbers are updated as a product moves from release to release, as well as for hardcopy updates within a given release.)
- When the softcopy version of a publication is updated for a new collection kit the order number it shares with the hardcopy version does not change. Also, the date in the edition notice remains that of the original publication. To compare softcopy with hardcopy, and softcopy with softcopy (on two editions of the collection kit, for example), check the last two characters of the publication's filename. The higher the number, the more recent the publication. For example, DFHPF104 is more recent than DFHPF103. Next to the publication titles in the CD-ROM booklet and the readme files, asterisks indicate publications that are new or changed.
- Updates to the softcopy are clearly marked by revision codes (usually a “#” character) to the left of the changes.

Index

Numerics

- 200 response
 - HTTP response 72
- 302 response
 - HTTP response 83
- 3270 applications 49
- 400 response
 - business logic interface 112
 - HTTP response 118, 119, 125
- 401 response
 - business logic interface 112
- 403 response
 - business logic interface 112
 - HTTP response 125
- 404 response
 - business logic interface 112
- 500 response
 - business logic interface 112
 - HTTP response 126
- 501 response
 - HTTP response 125, 126, 128
- 503 response
 - business logic interface 113

A

- absolute path in URL 69
- Accept-Encoding HTTP header 138
- Accept HTTP header 138
- Accept-Language HTTP header 138
- alias 81
- alias transaction CWBA 32
- analyzer 19, 37, 79
 - basic authentication sample 82
 - default 40, 119
 - designing and coding 37
 - programming reference 115
 - security sample 82
- Authorization HTTP header 138

B

- basic authentication analyzer 82
- basic authentication converter 82
- basic mapping support 17
- BMS 17
- business logic 17, 91

C

- character sets 149
- Charge-To HTTP header 138
- CICS business logic interface 17, 91
 - control flow for a program 94
 - control flow for a transaction 95
 - data flow for a program 96
 - data flow for a transaction(continue) 98
 - data flow for a transaction(start) 97

- CICS business logic interface 17, 91 (*continued*)
 - programming reference 107
- CICS Family: Client/Server Programming 93
- CICS program
 - designing and coding 69
- CICS system initialization parameters
 - TCPIP 29
 - XPCT 81
 - XPPT 81
 - XTRAN 81
 - XUSER 81
- CICS Web support 91
 - control flow for a program 19
 - control flow for a transaction 21
 - data flow for a program 22
 - data flow for a transaction (continue) 98
 - data flow for a transaction (start) 97
 - processing example 18
- CICSFOOT 52
- CICSHEAD 52
- client codepages 79, 149
- codepages 79, 149
- connection-oriented data transmission 8
- connectionless data transmission 8
- CONTENT_LANGUAGE environment variable 137
- Content-Language HTTP header 40, 137
- CONTENT_LENGTH environment variable 137
- Content-Length HTTP header 137
- CONTENT_TYPE environment variable 137
- Content-Type HTTP header 40, 137
- control flow
 - CICS business logic interface 94
 - CICS Web support 19
 - terminal oriented transaction 21, 95
- conversation token 141
- converter 43
 - basic authentication sample 82
 - designing and coding 43
 - programming reference 121
 - security sample 82
- converter name in URL 79
- CRLF 69, 118
- CWBA alias transaction 32
- CWXN Web attach transaction 19, 32

D

- data conversion 33
- data flow
 - CICS business logic interface 96
 - CICS Web support 22
 - terminal-oriented transaction (continue) 98
 - terminal-oriented transaction (start) 97
- datagram 8
- decode_client_address field 122
- decode_client_address_string field 122
- Decode converter function
 - designing and coding 44

Decode converter function (*continued*)
 programming reference 122
 decode_data_ptr field 122
 decode_eyecatcher field 123
 decode_function field 123
 decode_http_version_length field 123
 decode_http_version_ptr field 123
 decode_input_data_len field 43, 123
 decode_method_length field 123
 decode_method_ptr field 123
 decode_output_data_len field 123, 127
 decode_reason field 123
 decode_request_header_length field 123
 decode_request_header_ptr field 124
 decode_resource_length field 124
 decode_resource_ptr field 124
 decode_response field 124
 decode_server_program field 117, 124
 decode_user_data_length field 124
 decode_user_data_ptr field 124
 decode_user_token field 118, 124, 128
 decode_version field 125
 decode_volatile field 125, 128
 default URL 41, 79
 DFH\$WB1A 36, 76
 DFH\$WB1C 76
 DFH\$WBAU 82
 DFH\$WBSA 82
 DFH\$WBSB 82
 DFH\$WBSC 82
 DFH\$WBSN 36, 82
 DFH\$WBSN RDO group 30, 82
 DFH\$WBSR 141
 DFH\$WBST 141
 DFHCCNV 19
 DFHCCNV table 33
 DFHDHTXD 31
 DFHDHTXH 31
 DFHDHTXL 31
 DFHDHTXO 31
 DFHHTML DD name 30, 33
 DFHMDX macro 63
 DFHSIT 29
 DFHWBA alias program 33
 DFHWBADX 40, 119
 DFHWBBLI 107
 DFHWBENV (environment variables program) 36, 71,
 76, 137
 DFHWBEP, Web error program 47
 DFHWBHH conversion template 34
 DFHWBHH conversion template name 33
 DFHWBOUT macro 66
 DFHWBPA 143
 DFHWBTL 131
 DFHWBTTA 20, 49, 95
 DFHWBUD conversion template 34
 DFHWBUD conversion template name 34, 41
 DFHWBUD RDO group 30
 distributed application design 11
 distributed computing 6
 distributed transaction processing 7

DNS1 35
 dotted decimal 9
 double-byte character set (DBCS) 34
 DPL 6
 DPL subset 71

E

ECI 93
 ECI request
 processing example 94
 EDF 89
 Encode converter function
 designing and coding 45
 programming reference 127
 encode_data_ptr field 127
 encode_eyecatcher field 127
 encode_function field 127
 encode_input_data_len field 127
 encode_reason field 128
 encode_response field 128
 encode_user_token field 124, 128
 encode_version field 128
 ENTER TRACENUM command 89
 environment variables program (DFHWBENV) 36, 71,
 76, 137
 ephemeral port numbers 9
 EXCI 93
 EXCI request
 processing example 93
 EXEC CICS commands
 DOCUMENT 72
 DOCUMENT CREATE 78
 TCPIP 72
 WEB 72
 WEB ENDBROWSE FORMFIELD 70
 WEB ENDBROWSE HTTPHEADER 70
 WEB EXTRACT 70
 WEB READ FORMFIELD 70
 WEB READ HTTPHEADER 70
 WEB READNEXT FORMFIELD 70
 WEB READNEXT HTTPHEADER 70
 WEB RECEIVE 71, 78
 WEB SEND 78
 WEB STARTBROWSE FORMFIELD 70
 WEB STARTBROWSE HTTPHEADER 70
 WEB WRITE 72
 EXEC CICS LINK 93
 external call interface 93
 external CICS interface 93

F

File Transfer Protocol 9
 From HTTP header 138
 function shipping 7

H

HANDLE ABEND command 89
 hidden field in HTML form 141

- host name in URL 69
- HTML 17
- HTML form 69
- HTML template manager 81, 131
 - programming reference 132
 - setting up a sub-library 33
- HTML templates 57
- HTTP 17
- HTTP_ACCEPT_ENCODING environment variable 138
- HTTP_ACCEPT environment variable 138
- HTTP_ACCEPT_LANGUAGE environment variable 138
- HTTP_AUTHORIZATION environment variable 138
- HTTP_CHARGE_TO environment variable 138
- HTTP_FROM environment variable 138
- HTTP_IF_MODIFIED_SINCE environment variable 138
- HTTP method 69
- HTTP_PRAGMA environment variable 138
- HTTP_REFERER environment variable 139
- HTTP request 69
- HTTP request header 69
 - Accept 138
 - Accept-Encoding 138
 - Accept-Language 138
 - Authorization 138
 - Charge-To 138
 - Content-Language 40, 137
 - Content-Length 137
 - Content-Type 40, 137
 - From 138
 - If-Modified-Since 138
 - Keep-Alive 40
 - Pragma 138
 - Referer 139
 - User-Agent 139
- HTTP response 71
- HTTP response codes 72
- HTTP response header 72
- HTTP_USER_AGENT environment variable 139
- HTTP user data 70
- HTTP version 69
- hypertext markup language 17
- hypertext transfer protocol 17

I

- IANA character set 149
- IBM CCSID character set 149
- If-Modified-Since HTTP header 138
- INFOANA 88
- internet address 8
- Internet Protocol (IP) 8
- ISO 3316 language code 137
- ISO 639 country code 137
- ISO 8859-1 character set 33

K

- Keep—Alive header 40

L

- Latin-1 character set 33
- lightpen operation 54
- limitations of Web 3270 support 62
- load modules 31

M

- messages and codes 87

N

- name server 35
- non-HTTP requests 20

P

- parser program 143
- persistent connections 40
- port number 8
- port number in URL 69
- port numbers 35
- Pragma HTTP header 138
- presentation logic 17, 91
- processing examples
 - CICS Web support 18
 - ECI request 94
 - EXCI request 93
- PROGRAM definitions 33
- programming models 9
- pseudoconversational model 10

Q

- QR TCB 86
- QUERY_STRING environment variable 137
- query string in URL 69, 141

R

- Referer HTTP header 139
- REMOTE_ADDR environment variable 137
- REMOTE_HOST environment variable 137
- REMOTE_USER environment variable 137
- REQUEST_METHOD environment variable 138
- requester types 17, 93
- RP TCB 86

S

- samples
 - application 36, 76
 - basic authentication analyzer 82
 - basic authentication converter 82
 - security analyzer 82
 - security converter 82
 - sign-on program 82
 - state management program 141
- security 81
- security analyzer 82

- security converter 82
- security support 7
- selector pen operation 54
- SERVER_NAME environment variable 138
- SERVER_PORT environment variable 138
- SERVER_PROTOCOL environment variable 138
- SERVER_SOFTWARE environment variable 138
- service types 17
- session token 141
- sign-on sample program 82
- sockets interface 8
- state management sample program 82, 141
- sub-library 30
- symbol list 74
- symbols in an HTML template 74

T

- task control blocks 86
- TCP/IP 7
- TCP/IP port in URL 69
- TCP62 94
- TCPIP system initialization parameter 29
- TCPIPSERVICE definition 31, 35
- TD queue 31
- Telnet 9
- temporary storage queue 30
- tools
 - environment variables program 137
 - HTML template manager 81, 131
 - parser program 143
- transaction routing 7
- transient data queue 31
- Transmission Control Protocol (TCP) 8
- TS queue 30

U

- uniform resource locator 69
- URL 69, 79
 - absolute path 69
 - host name 69
 - port number 69
 - query string 69
- URL, default 79
- URP_DISASTER response
 - in analyzer 118
 - in Decode 126
 - in Encode 128
- URP_EXCEPTION response
 - in analyzer 118
 - in Decode 125
- URP_INVALID response
 - in analyzer 118
 - in Decode 125
- URP_OK_LOOP 128
- URP_OK response
 - in analyzer 118
 - in Decode 125
 - in Encode 128
- URP_RECEIVE_OUTSTANDING reason code 118
- User-Agent HTTP header 139

- user data 70
- User Datagram Protocol (UDP) 8
- user-replaceable programs 33

V

- VSE/ESA sub-library 30

W

- wbbl_client_address 109
- wbbl_client_address_length 109
- wbbl_client_address_string 109
- wbbl_converter_program_name 109
- wbbl_eyecatcher field 109
- wbbl_header_length 109
- wbbl_header_offset 110
- wbbl_http_version_length 110
- wbbl_http_version_offset 110
- wbbl_indata_length 110
- wbbl_indata_offset 110
- wbbl_indata_ptr 110
- wbbl_length field 110
- wbbl_method_length 110
- wbbl_method_offset 110
- wbbl_mode field 110
- wbbl_outdata_length 111
- wbbl_outdata_offset 111
- wbbl_outdata_ptr 111
- wbbl_prolog_size 111
- wbbl_resource_length 111
- wbbl_resource_offset 111
- wbbl_response field 111
- wbbl_server_program_name 111
- wbbl_status_size 111
- wbbl_user_data_length 111
- wbbl_user_token 112
- wbbl_vector_size 112
- wbbl_version field 112
- wbep_abend_code 145
- wbep_analyzer_reason 145
- wbep_analyzer_response 145
- wbep_client_address 145
- wbep_client_address_len 145
- wbep_converter_program 145
- wbep_converter_reason 145
- wbep_converter_response 145
- wbep_error_code 146
- wbep_eyecatcher 146
- wbep_failing_program 146
- wbep_http_response_code 146
- wbep_length 146
- wbep_message_len 146
- wbep_message_number 146
- wbep_message_ptr 146
- wbep_response_len 146
- wbep_response_ptr 146
- wbep_server_address 146
- wbep_server_address_len 146
- wbep_target_program 146
- wbep_tcpip_service_name 147
- wbep_version 147

wbra_alias_termid field 89
wbra_alias_tranid field 116
wbra_client_ip_address field 116
wbra_content_length field 116
wbra_converter_program field 116
wbra_dfhcnv_key field 116
wbra_eyecatcher field 116
wbra_function field 116
wbra_http_version_length field 116
wbra_http_version_ptr field 116
wbra_method_length field 116
wbra_method_ptr field 116
wbra_reason field 117
wbra_request_header_length field 117
wbra_request_header_ptr field 117
wbra_request_type field 117
wbra_resource_length field 117
wbra_resource_ptr field 117
wbra_response field 117
wbra_server_ip_address field 117
wbra_server_program field 117, 124
wbra_unescape 117
wbra_user_data_length field 118
wbra_user_data_ptr field 118
wbra_user_token field 118, 124
wbra_userid field 81, 118
wbtI_connect_token field 133
WBTL_DISASTER response 135
WBTL_EXCEPTION response 134
WBTL_FREEMAIN_ERROR reason 135
wbtI_function field 132
WBTL_GETMAIN_ERROR reason 135
wbtI_html_buffer_len field 134
wbtI_html_buffer_ptr field 134
WBTL_INVALID_BUFFER_PTR reason 135
WBTL_INVALID_FUNCTION reason 135
WBTL_INVALID response 135
WBTL_INVALID_SYMBOL_LIST reason 135
WBTL_INVALID_TOKEN reason 135
WBTL_OK response 134
WBTL_PAGE_TRUNCATED reason 135
wbtI_reason field 133
wbtI_response field 133
wbtI_symbol_list_len field 134
wbtI_symbol_list_ptr field 134
wbtI_template_abstime field 133
wbtI_template_buffer_len field 134
wbtI_template_buffer_ptr field 133
wbtI_template_name field 133
WBTL_TEMPLATE_NOT_FOUND reason 135
WBTL_TEMPLATE_TRUNCATED reason 135
wbtI_version_no field 132
Web attach transaction CWXN 19, 32
Web error program, DFHWBEP 47
well-known ports 9
WRITEQ TD command 89

XUSER system initialization parameter 81

X

XPCT system initialization parameter 81
XPPT system initialization parameter 81
XTRAN system initialization parameter 81

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

Information Development Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-870229
 - From within the U.K., use 01962-870229
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink™ : HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



Program Number: 5648-054



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5765-00



Spine information:



CICS TS for VSE/ESA

Internet Guide

Release 1