CICS® Transaction Server for OS/390®

CICS Internet Guide

Release 3

SC34-5445-00

CICS® Transaction Server for OS/390®

CICS Internet Guide

Release 3

SC34-5445-00

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page iii.

First Edition (March 1999)

This edition replaces and makes obsolete the previous edition, SC33–1944. The technical changes for this edition are summarized under “Summary of changes” and are indicated by a vertical bar to the left of a change.

This book is based on the *CICS Internet and External Interfaces Guide*, SC33–1944, which remains current for CICS Transaction Server for OS/390 Release 2.

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.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Programming interface information

This book is intended to help you use the external interfaces provided by the CICS Transaction Server for OS/390. 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

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AD/Cycle
BookManager
C/370
CICS
CICS/ESA
CICS/MVS
DB2
DFS
Enterprise Systems Architecture/390
IBM
IMS
Language Environment
MQ
MQSeries
MVS/ESA
OpenEdition
OS/2
OS/390
RACF
RT
SAA
System/390

VTAM WebExplorer

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.

Other company, product, and service names may be trademarks or service marks of others.

Preface

What this book is about

This book describes how you can make the CICS® transaction processing services of CICS TS for OS/390® available to a variety of Internet users and TCP/IP-based applications.

How to use this book

This book is intended to complement the *CICS External Interfaces Guide* and 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

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 OS/390.

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

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 usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

Subsequent updates will probably be available in softcopy before they are available in hardcopy. This means that at any time from the availability of a release, softcopy versions should be regarded as the most up-to-date.

For CICS Transaction Server books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-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.

Updates to the softcopy are clearly marked by revision codes (usually a "#" character) to the left of the changes.

Contents

Bibliography

CICS Transaction Server for OS/390

CICS Transaction Server for OS/390: Planning for Installation	GC33-1789
CICS Transaction Server for OS/390 Release Guide	GC34-5352
CICS Transaction Server for OS/390 Migration Guide	GC34-5353
CICS Transaction Server for OS/390 Installation Guide	GC33-1681
CICS Transaction Server for OS/390 Program Directory	GI10-2506
CICS Transaction Server for OS/390 Licensed Program Specification	GC33-1707

CICS books for CICS Transaction Server for OS/390

General	
CICS Master Index	SC33-1704
CICS User's Handbook	SX33-6104
CICS Transaction Server for OS/390 Glossary (softcopy only)	GC33-1705
Administration	
CICS System Definition Guide	SC33-1682
CICS Customization Guide	SC33-1683
CICS Resource Definition Guide	SC33-1684
CICS Operations and Utilities Guide	SC33-1685
CICS Supplied Transactions	SC33-1686
Programming	
CICS Application Programming Guide	SC33-1687
CICS Application Programming Reference	SC33-1688
CICS System Programming Reference	SC33-1689
CICS Front End Programming Interface User's Guide	SC33-1692
CICS C++ OO Class Libraries	SC34-5455
CICS Distributed Transaction Programming Guide	SC33-1691
CICS Business Transaction Services	SC34-5268
Diagnosis	
CICS Problem Determination Guide	GC33-1693
CICS Messages and Codes	GC33-1694
CICS Diagnosis Reference	LY33-6088
CICS Data Areas	LY33-6089
CICS Trace Entries	SC34-5446
CICS Supplementary Data Areas	LY33-6090
Communication	
CICS Intercommunication Guide	SC33-1695
CICS Family: Interproduct Communication	SC33-0824
CICS Family: Communicating from CICS on System/390	SC33-1697
CICS External Interfaces Guide	SC33-1944
CICS Internet Guide	SC34-5445
Special topics	
CICS Recovery and Restart Guide	SC33-1698
CICS Performance Guide	SC33-1699
CICS IMS Database Control Guide	SC33-1700
CICS RACF Security Guide	SC33-1701
CICS Shared Data Tables Guide	SC33-1702
CICS Transaction Affinities Utility Guide	SC33-1777
CICS DB2 Guide	SC33-1939

CICSPlex SM books for CICS Transaction Server for OS/390

General	
CICSPlex SM Master Index	SC33-1812
CICSPlex SM Concepts and Planning	GC33-0786
CICSPlex SM User Interface Guide	SC33-0788
CICSPlex SM View Commands Reference Summary	SX33-6099
Administration and Management	
CICSPlex SM Administration	SC34-5401
CICSPlex SM Operations Views Reference	SC33-0789
CICSPlex SM Monitor Views Reference	SC34-5402
CICSPlex SM Managing Workloads	SC33-1807
CICSPlex SM Managing Resource Usage	SC33-1808
CICSPlex SM Managing Business Applications	SC33-1809
Programming	
CICSPlex SM Application Programming Guide	SC34-5457
CICSPlex SM Application Programming Reference	SC34-5458
Diagnosis	
CICSPlex SM Resource Tables Reference	SC33-1220
CICSPlex SM Messages and Codes	GC33-0790
CICSPlex SM Problem Determination	GC33-0791

Other CICS books

CICS Application Programming Primer (VS COBOL II)	SC33-0674
CICS Application Migration Aid Guide	SC33-0768
CICS Family: API Structure	SC33-1007
CICS Family: Client/Server Programming	SC33-1435
CICS Family: General Information	GC33-0155
CICS 4.1 Sample Applications Guide	SC33-1173
CICS/ESA 3.3 XRF Guide	SC33-0661

If you have any questions about the CICS Transaction Server for OS/390 library, see *CICS Transaction Server for OS/390: Planning for Installation* which discusses both hardcopy and softcopy books and the ways that the books can be ordered.

Non-CICS books

OS/390 UNIX System Services

- *OS/390 UNIX System Services User's Guide*, SC28-1891
- *OS/390 UNIX System Services Command Reference*, SC28-1892
- *OS/390 UNIX System Services Programming Tools*, SC28-1904
- *OS/390 UNIX System Services Messages and Codes*, SC28-1908
- *OS/390 UNIX System Services Programming: Assembler Callable Services Reference*, SC28-1899
- *OS/390 UNIX System Services File System Interface Reference*, SC28-1909
- *OS/390 Using REXX and OS/390 UNIX System Services*, SC28-1905
- *OS/390 UNIX System Services Communications Server Guide*, SC28-1906
- *OS/390 UNIX System Services Parallel Environment: MPI Programming and Subroutine Reference*, SC33-6696

OS/390 eNetwork Communications Server

The OS/390 eNetwork Communications Server library is as follows:

- *OS/390 eNetwork Communications Server: IP Configuration Guide*, SC31-8513
- *OS/390 eNetwork Communications Server: IP Planning and Migration Guide*, SC31-8512
- *OS/390 eNetwork Communications Server: IP CICS Sockets Guide*, SC31-8518
- *OS/390 eNetwork Communications Server: IP Application Programming Interface Guide*, SC31-8516
- *OS/390 eNetwork Communications Server: IP Programmer's Reference*, SC31-8515
- *OS/390 eNetwork Communications Server: IP User's Guide*, GC31-8514
- *OS/390 eNetwork Communications Server: Quick Reference*, SX75-0121
- *OS/390 eNetwork Communications Server: IP Diagnosis*, SC31-8521
- *OS/390 eNetwork Communications Server: High Speed Access Services*, GC31-8676

Language Environment

- *OS/390 Language Environment Programming Guide*, SC28-1939
- *OS/390 Language Environment Programming Reference*, SC28-1940
- *OS/390 Language Environment Customization*, SC28-1941

Miscellaneous

The following publications contain related information:

- *CICS 4.1 Sample Applications Guide*, SC33-1173
- *Accessing CICS Business Applications from the World Wide Web*, SG24-4547
- *IBM Internet Connection Server for MVS/ESA Up and Running!*, SC31-8204
- *How to Secure the Internet Connection Server for MVS/ESA*, SG324-4803
- *OS/390 Internet BonusPak*, G221-9001
- *IBM's Official Guide to Building a Better Web Site*, SR23-7270

- CICS SupportPacs, at:
  `http://www.software.ibm.com/ts/cics/txppacs/txpc1.htm#int`

Information about Java can be found at:
`http://www.javasoft.com/`

Information on the World Wide Web

Information about the hypertext transfer protocol (HTTP), the hypertext markup language (HTML), CORBA, and secure sockets layer (SSL) is to be found 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. For HTML information, consult the following:

- *Hypertext Markup Language (HTML)*
  `http://www.w3.org/pub/WWW/MarkUp/`
- *HTML reference manual*
  `http://www.sandia.gov/sci_compute/html_ref.html`
- *Hypertext Markup Language - 2.0*
  `http://ds.internic.net/rfc/rfc1866.txt`
- *HTML, the complete guide*
  `http://www.emerson.emory.edu/services/html/html.html`
- *Introducing HTML 3.2*
  `http://www.w3.org/pub/WWW/MarkUp/Wilbur/`
- *Working draft of HTML 4.0*
  `http://www.w3.org/TR/WD-html40-970708/`

Secure sockets layer (SSL)

For SSL information, consult the following:

- *Overview of SSL*
  `http://www.netscape.com/newsref/ref/rsa.html`
- *Description of Public-key Cryptography Standards*
  `http://www.rsa.com/rsalabs/pubs/PKCS`
- *Description of asymmetric encryption*
  `http://www.rsa.com/rsa/rsamath/`
- *The ITU-T X.509 recommendation for certificates*
  `http://www.itu.org/itudoc/itu-t/rec/x/x500up/x509_27505.html`

CORBA

CICS supports CORBA 2.0 and IIOP 1.0. The following URL contains CORBA architecture information:

`http://www.omg.org/library`

Summary of changes

This book is based on the *CICS Internet and External Interfaces Guide* for CICS Transaction Server for OS/390 Release 2. Changes from that edition are marked by vertical lines to the left of the changes.

Changes for this edition

The major changes to CICS that affect this book are:

- The addition of information about IIOP inbound to Java.
- The addition of information about the EXEC CICS DOCUMENT commands.
- The addition of information about the EXEC CICS WEB commands.
- The addition of information about secure sockets layer.
- The addition of information about the Web error program, DFHWBEP.
- The addition of information about the TCPIPSERVICE resource definition.
- The addition of information about the DOCTEMPLATE resource definition.
- The business logic interface has changed its name from DFHWBA1 to DFHWBBLI, and its parameters have changed from **wba1**_ to **wbbl**_.

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:

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.

IBM Websphere
This is an MVS application that supports Web browsers and routes their requests into CICS.

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.

CORBA clients
CICS provides support for inbound IIOP requests for CICS Java applications.

JVM applications
Java Virtual Machine applications can use a local gateway connection that uses the EXCI to pass requests to CICS.

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.

CICS Transaction Gateway for OS/390
This is a version of the CICS Transaction Gateway that runs on OS/390, and uses the CICS external CICS interface (EXCI) to pass requests to CICS. The CICS Transaction Gateway for OS/390 supports the use of ECI requests, but not EPI requests.

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 TS.

User socket applications
User socket applications can use the CICS Sockets feature of CICS TS. See the *OS/390 eNetwork Communications Server: IP CICS Sockets Guide*.

MQSeries® users
MQSeries users can use the 3270 CICS bridge to access CICS transactions. See See the *CICS External Interfaces Guide* for information.

MVS™ applications
Applications running in MVS address spaces 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*.

DCE RPC clients
DCE RPC clients use the Application Support (AS) server to access CICS programs. See the *CICS External Interfaces Guide*.

ONC RPC clients
ONC RPC clients can use CICS ONC RPC support to access CICS programs. See the *CICS External Interfaces Guide* for information about ONC PRC.

Telnet clients
Telnet clients can use TN3270 to start transactions. See the *OS/390 eNetwork Communications Server: IP Configuration 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 TS. Programs running in CICS TS can use the CICS front end programming interface (FEPI) to start transactions in the same or another instance of CICS TS. 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.

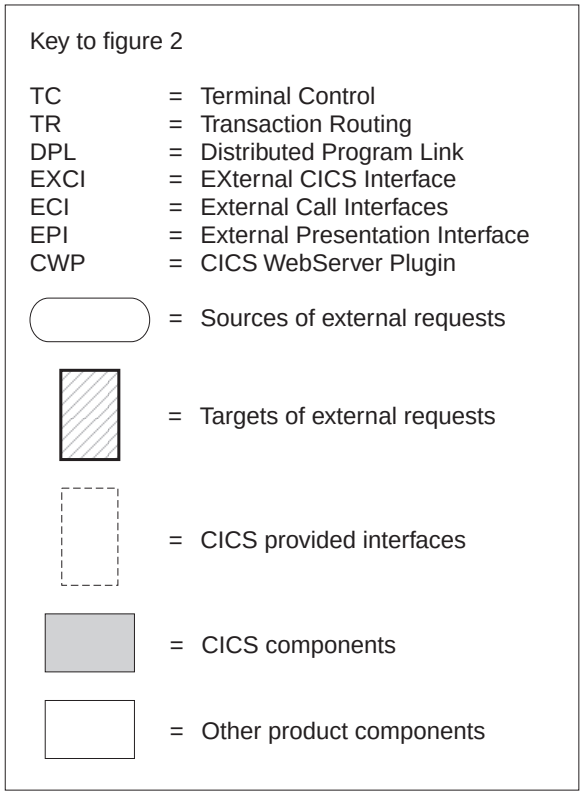


Figure 1.

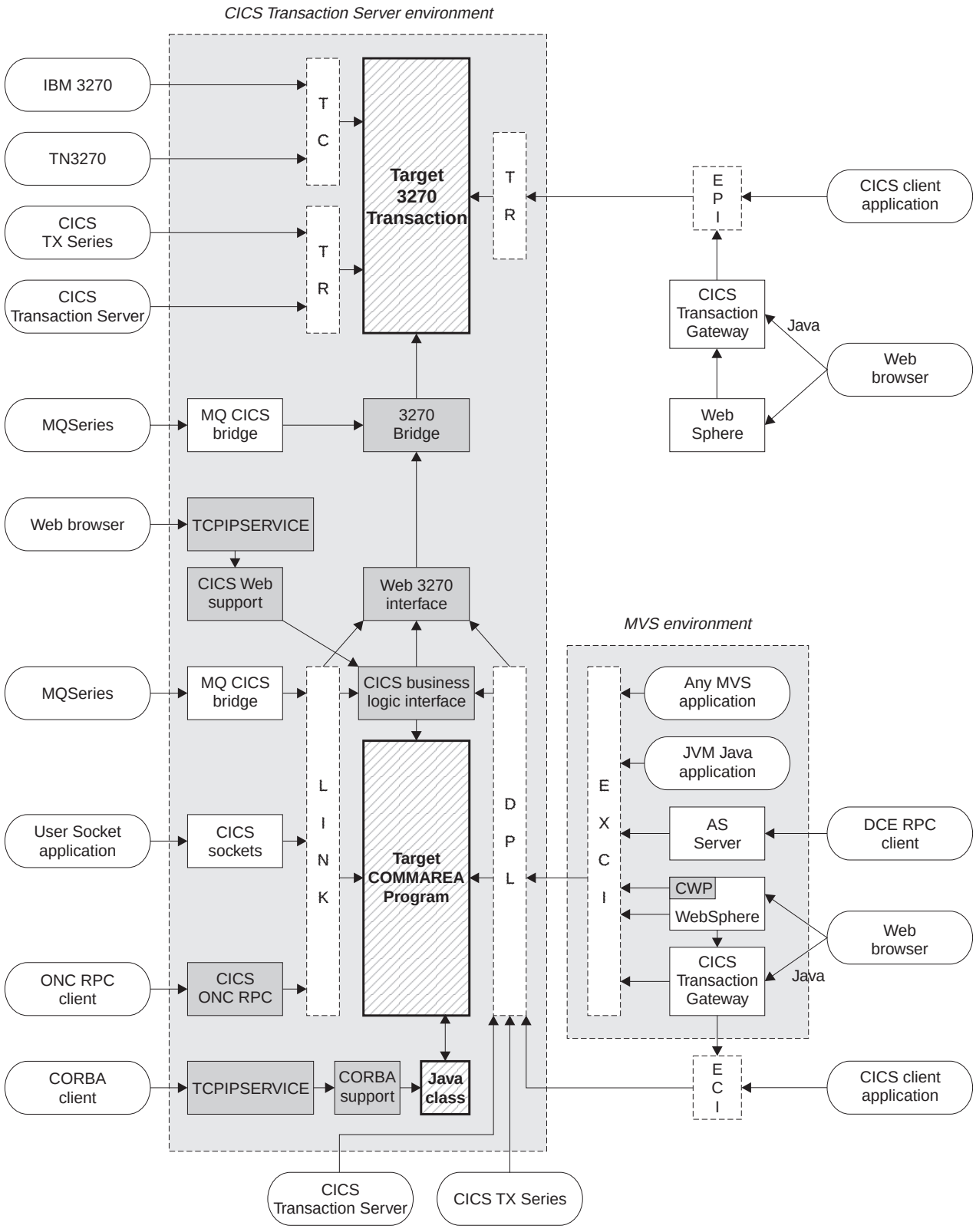


Figure 2. Client access to existing business logic

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:

Internet Inter-Orb Protocol (IIOP)
CORBA clients can access CICS Java servers using IIOP.

Distributed Computing Environment (DCE)
The remote procedure call model implemented by the Open Software Foundation's DCE is supported in CICS.

Distributed program link (DPL)
This is similar to a DCE remote procedure call. 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)
An MVS client program links to a CICS server program. Again, this is similar to a DCE RPC.

The external call interface (ECI)
The ECI enables CICS Transaction Server for OS/390 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 OS/390 supports:

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

RACF or an equivalent security manager provides mechanisms similar to the DCE access control lists and login facility.

There is no CICS concept similar to the DCE Directory Service. 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 shows the TCP/IP protocols used by CICS ONC RPC 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.

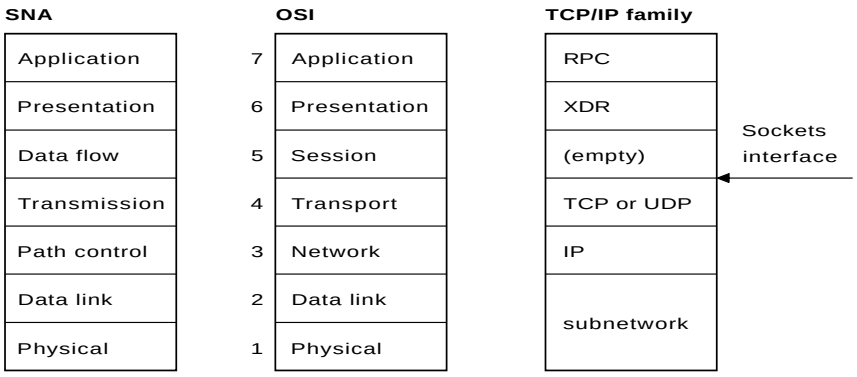


Figure 3. TCP/IP protocols compared to the OSI and SNA models

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 that 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) 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.

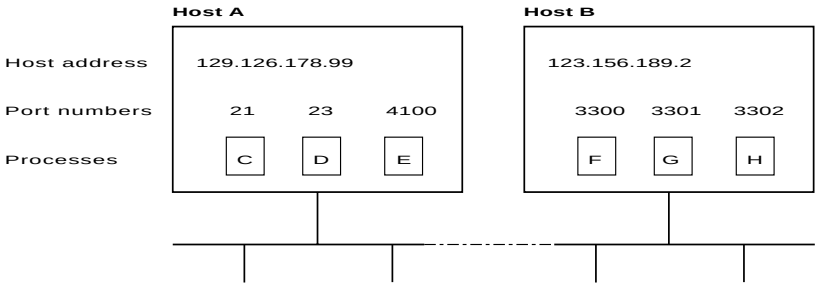


Figure 4. How applications are addressed

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 on page 9, 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 section 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, CORBA, Java). This affects the transport and presentation protocol that you use, and may affect your decision on whether to use secure sockets layer (SSL).

Accessing CICS from the Web

CICS Web support allows you to use a Web browser as a graphical user interface for business logic applications. 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 OS/390, or a 3–tier solution, where the Web server is external to OS/390 (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 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 12. Writing CICS programs to process HTTP requests” on page 71 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 two Java environments;

- Java support provided by the CICS Transaction Gateway for OS/390
- and inbound IIOP support of CORBA clients

This section outlines the differences between them.

CICS Transaction Gateway for OS/390

The Java language can be used to construct Java applets and Java applications, both of which are used in the CICS Transaction Gateway for OS/390. 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 writen for the Java gateway would use the ibm.cics.jgate.client.ECIRequest class to produce an EXCI call to communicate with a COBOL program using a commarea.

Inbound IIOP support of CORBA clients

When CICS receives an IIOP request from a CORBA client (using the same listener as CICS Web support), the request is processed in a Java environment within CICS. In this environment, Java programs execute using JCICS classes as the CICS application programming interface. For example, a Java class invoked by a CORBA client results in an object being called in CICS that in turn may execute a JCICS API request to do the equivalent of an EXEC CICS LINK. (The JCICS Java API is defined in the Javadoc HTML provided in *dfjcics_docs.zip*, downloaded during CICS installation.). The CORBA client support, which runs this Java environment inside CICS, offers the following benefits:

- function encapsulation, enabling rapid reuse of applications
- input and output data formatting when translation code is generated
- seamless integration with application data types, resulting in strong typing if there are no coding errors in the input and output data
- the use of standard IIOP protocol provides client autonomy by means of a vendor-independent client side
- object-oriented and procedural applications can co-exist in the same region, providing seamless access to CICS services and existing applications.

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 23, 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 13 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 transaction reads the data with the EXEC CICS RECEIVE MAP command, updates the record in the file, and sends the response with the EXEC CICS SEND MAP command.

The EXEC CICS RECEIVE and EXEC CICS SEND MAP commands are part of the transaction’s presentation logic, while the EXEC CICS READ UPDATE and EXEC CICS REWRITE commands are part of the business logic.

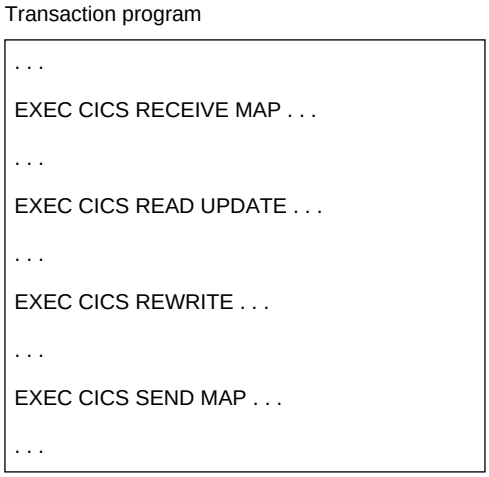


Figure 5. CICS functions in a single application program

A sound principle of modular programming in CICS application design is to separate the presentation logic from the business logic, and to use a communication area and the EXEC CICS LINK command to make them into a single transaction. Figure 6 illustrates this approach to application design.

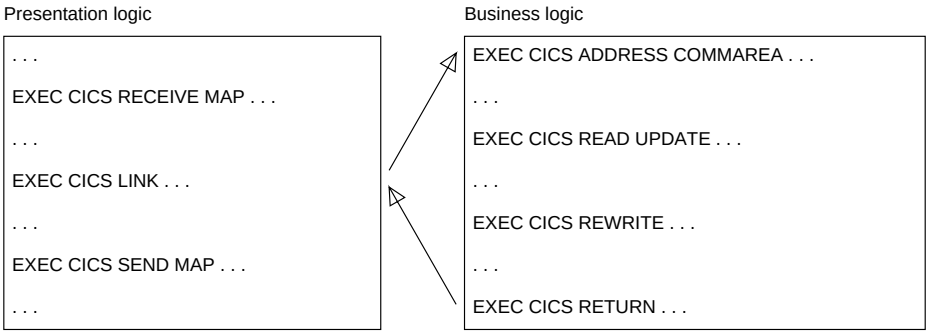


Figure 6. Separation of business and presentation logic

Once the business logic of a transaction has been isolated from the presentation logic and given a communication area interface, it is available for reuse with different presentation methods. For example, you could use CICS Web support with the CICS business logic interface, to implement a two-tier model where the presentation logic is HTTP-based.

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 17 describes support for web browsers through CICS Web support and through the IBM WebSphere Application Server for OS/390.
- “Part 3. The CICS business logic interface” on page 93 describes the CICS business logic interface
- “Part 4. Using secure sockets layer (SSL)” on page 105 describes the secure sockets layer (SSL).
- “Part 5. Using the CICS Transaction Gateway for OS/390” on page 115 describes the CICS Transaction Gateway for OS/390, and the CICS Java classes.
- “Part 6. CORBA client support” on page 165 describes the IIOP (Internet Inter-orb Protocol).

Part 2. CICS Web support

This part of the book describes CICS Web support.

It contains:

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:

- CICS Web support 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.

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 12.

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

"Chapter 4. Planning for CICS Web support" on page 27 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:

1. 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.
2. Web browsers that are connected to the IBM WebSphere Application Server for OS/390. A CICS-provided WebServer Plugin that operates within the IBM WebSphere Application Server for OS/390 uses user-provided definitions to relate the HTTP request to the required CICS transaction processing services. The CICS business logic interface services the request.
3. Non-HTTP clients — see "Dealing with non-HTTP requests" on page 23.

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.

Processing examples

Figure 7 shows how CICS Web support processes a request from a Web browser that is connected to OS/390 eNetwork Communications Server.

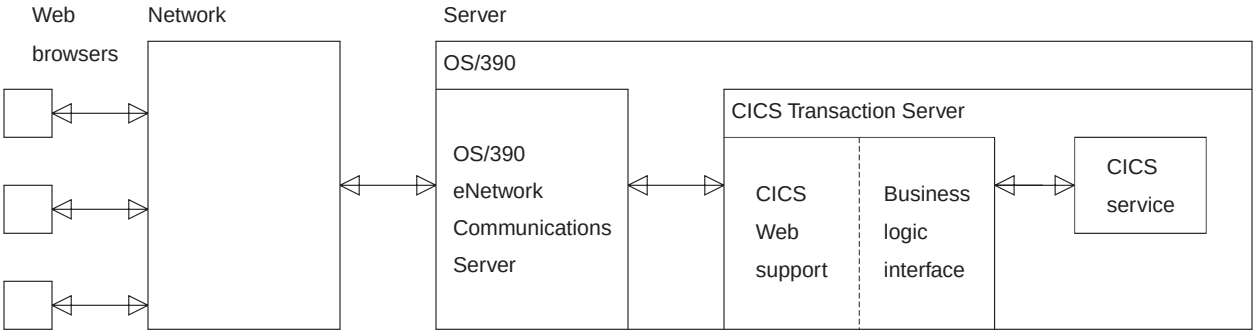


Figure 7. Processing a request to CICS Web support

The Web browser is an HTTP client. It constructs an HTTP request, which is passed across the network to OS/390 eNetwork Communications Server in the server. OS/390 eNetwork Communications Server 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.

Figure 8 on page 21 shows how the CICS Web support processes a request from a Web browser that is connected to the IBM WebSphere Application Server for OS/390.

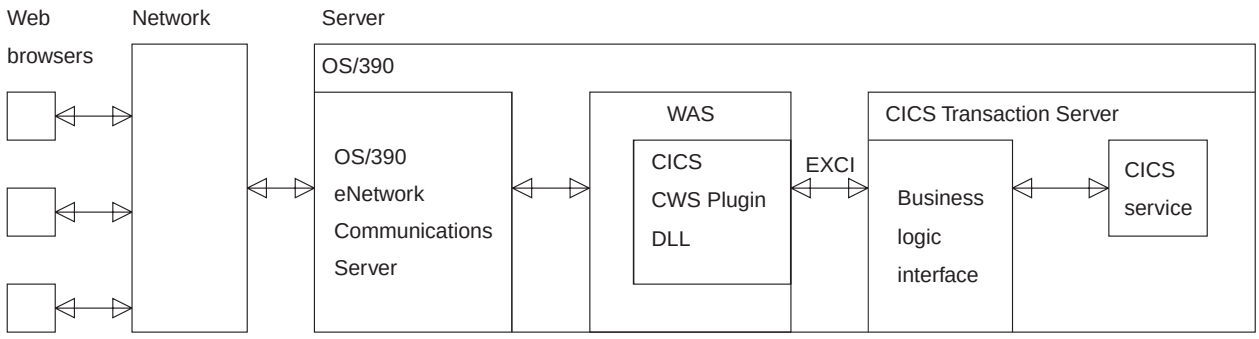


Figure 8. Processing a request from the IBM WebSphere Application Server for OS/390

The Web browser constructs an HTTP request which is passed across the network to OS/390 eNetwork Communications Server in the server. OS/390 eNetwork Communications Server relays the request to the IBM WebSphere Application Server for OS/390, which uses the CICS WebServer Plugin (CICS-provided code and user-provided definitions) to construct a request for the CICS business logic interface. The CICS business logic interface ensures that the CICS TS provides the requested service, and returns any output in the communication area.

Control flow in request processing

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

Using CICS Web support to call a program

Figure 9 on page 22 shows the control flow through CICS Web support to a CICS program.

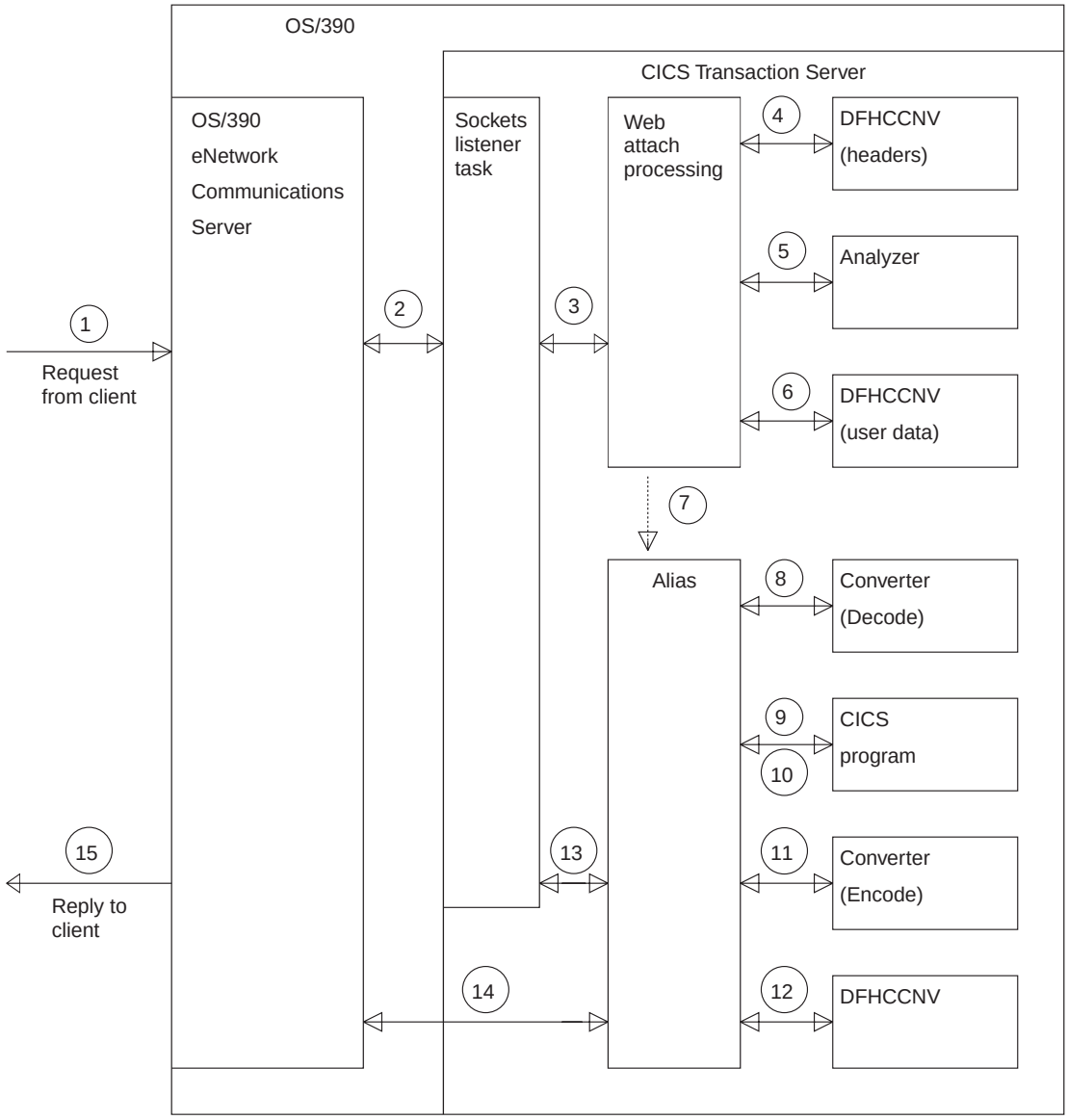


Figure 9. Calling a program with CICS Web support—control flow

1. An HTTP request arrives in OS/390 eNetwork Communications Server from a Web browser.
2. The Sockets listener task monitors the OS/390 eNetwork Communications Server interface for incoming HTTP requests.
3. The Sockets listener task attaches Web attach transaction CWXN. CWXN or its alias should be specified as the TRANSACTION on the TCPIPSERVICE 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.
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 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.
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 OS/390 eNetwork Communications Server 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 9 on page 22 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 10 on page 24 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.

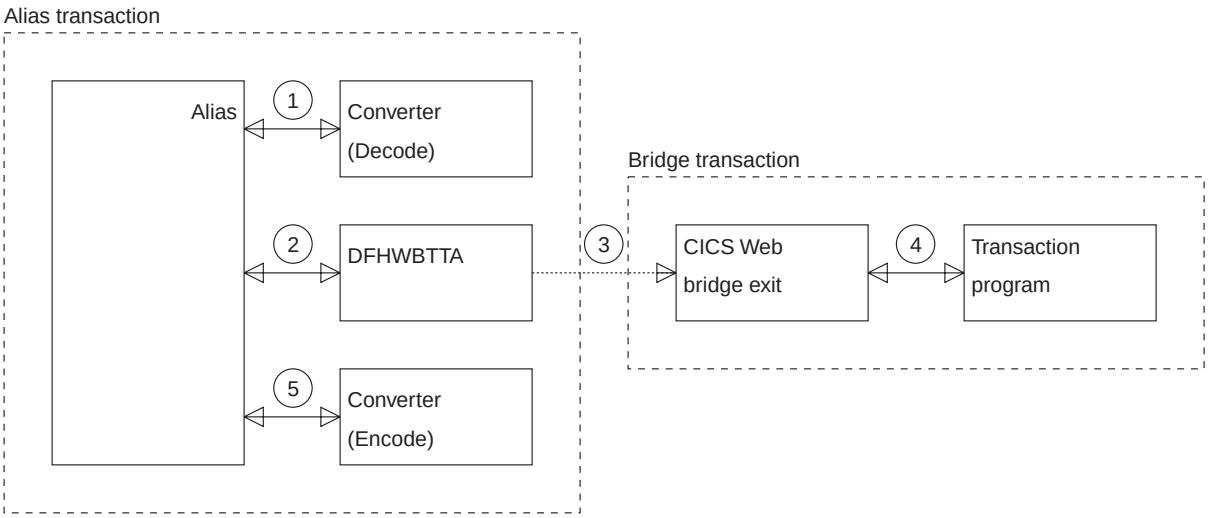


Figure 10. Running a transaction with CICS Web support—control flow

1. If the analyzer requests a converter, the alias calls it, requesting the **Decode** function. **Decode** sets up the communication area for DFHWBTTA.
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 you will use, and how you will 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 11 on page 25 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 21, some of these steps are optional. See “Chapter 12. Writing CICS programs to process HTTP requests” on page 71 for more information about HTTP headers and HTTP requests.

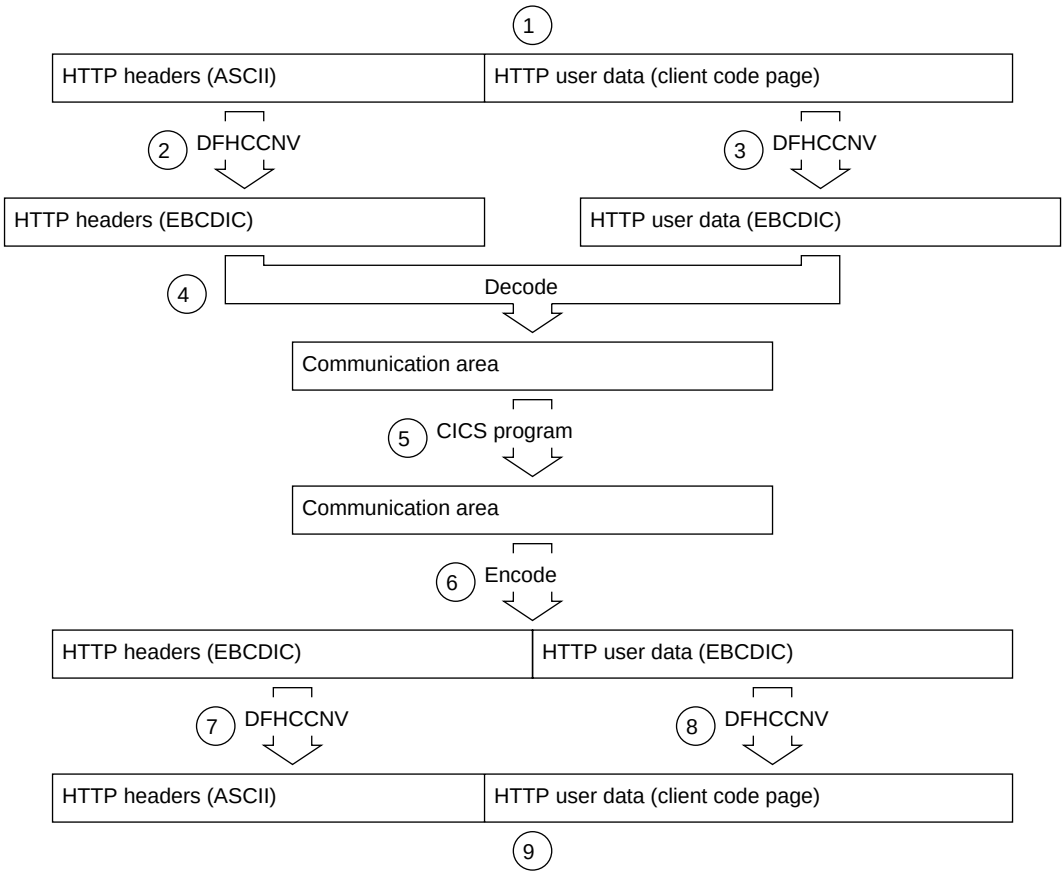


Figure 11. Calling a program using the CICS Web support commarea method—data flow

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 “HTTP responses” on page 73.
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 is a task-oriented guide to setting up CICS Web support. It is organized as follows:

- "Planning tasks" describes the planning tasks. 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 task planning.
- "Set-up tasks" on page 28
- "Programming tasks for CICS Web support" on page 28
- "Programming tasks for client systems" on page 29
- "Operations tasks" on page 29
- "Problem determination tasks" on page 29

Planning tasks

- Decide which CICS transaction processing services are to be made available to users of CICS Web support and the CICS business logic interface. These services might be CICS programs, or CICS transactions.
- Decide which CICS resources are to be accessed by which TCPIPSERVICE definitions. See the *CICS Resource Definition Guide* for details of the TCPIPSERVICE resource definition.
- Decide which services require security.
- Decide which TCP/IP ports CICS is to listen on.
- Decide which TCP/IP stack CICS is to register with.
- Decide which users are to be allowed to use which services.
- Specify the URLs that are to be used to give access to the services.
  - If requests are received by CICS Web support, the decision about URLs will affect the specification of the analyzer. "The default analyzer" on page 46 describes the conventions accepted by the default analyzer supplied with CICS Web support.
  - If HTTP requests are from the IBM WebSphere Application Server for OS/390, the decision about URLs will affect the configuration statements that you supply to the IBM WebSphere Application Server for OS/390. "Chapter 6. The CICS WebServer Plugin" on page 41 describes the mapping of URLs from browsers into CICS transaction processing services.
  - 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 197 describes the communication area that callers must supply, and explains what the CICS business logic interface does with its input.
- For CICS transactions that use BMS, decide what customization of the HTML output is necessary. See "Chapter 11. Creating HTML templates from BMS definitions" on page 61.
- To use secure sockets layer (SSL), you must build a key database and obtain a server certificate. See "Part 4. Using secure sockets layer (SSL)" on page 105.

Set-up tasks

- Install any prerequisite software and fixes. See “Prerequisites for using CICS Web support” on page 31.
- If you have an assembled temporary storage table (TST) that does not specify MIGRATE=YES and that has not been migrated to RDO, message DFHAM4895 is issued during CICS initialization. This means that the installation of the default TSMODEL has failed, and the TST is used to determine the attributes of any queues created by CICS Web support. To avoid this, either migrate TST definitions to RDO and specify TST=NO in the system initialization table, or re-assemble the existing TST with MIGRATE=YES.
- Write extra definitions for alias transactions for CICS Web support. See “TRANSACTION definitions for extra alias transactions” on page 35.
- Write definitions for CICS Web support TCPIPSERVICEs.
- Set up a PDS for the template manager. See “Setting up a PDS for the template manager” on page 36.
- Define a conversion table for CICS Web support. See “Defining a conversion table” on page 36.
- Write definitions of the user-replaceable programs. See “PROGRAM definitions for user-replaceable programs” on page 36.
- Write configuration statements for the IBM WebSphere Application Server for OS/390. See “Chapter 6. The CICS WebServer Plugin” on page 41.
- Put appropriate definitions in the external security manager for the template manager dataset. See “Security for the CICS Web support” on page 83
- Put appropriate definitions in the external security manager for CICS Web support transactions. See “Security for the CICS Web support” on page 83.
- If you want CICS to access a dynamic name server other than the system default (for example, a dynamic name server configured for connection optimisation), you must set up the SYSTCPD DD statement. See“The OS/390 eNetwork Communications Server” on page 38.

Programming tasks for CICS Web support

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).

- Write analyzers for CICS Web support for requests for CICS programs and for CICS transactions. See “Chapter 7. Writing an analyzer for CICS Web support” on page 43, and “Appendix B. Reference information for DFHWBADX” on page 205.
- Write converters for CICS Web support and the CICS business logic interface for requests for CICS programs and for CICS transactions. See “Chapter 8. Writing a converter” on page 49, and “Appendix C. Reference information for the converter” on page 211
- Write CICS application programs that can process HTTP and HTML either in the communication area or using the EXEC CICS WEB commands. See “Chapter 12. Writing CICS programs to process HTTP requests” on page 71.

- Write customization macros for converting between HTML and BMS. See “Chapter 11. Creating HTML templates from BMS definitions” on page 61.

Programming tasks for client systems

- Write MVS applications to use the EXCI to communicate with the CICS business logic interface. There will be 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 197.
- Write workstation applications to use the ECI to communicate with the CICS business logic interface. There will be 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 197.
- Write CICS applications to use EXEC CICS LINK to communicate with the CICS business logic interface. There will be 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 197.

Operations tasks

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

  and CEDA for these resource types:
  - TCPIPSERVICE
  - DOCTEMPLATE

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

Problem determination tasks

- Use CICS problem determination aids to analyze problems with CICS Web support and the CICS business logic interface. See “Chapter 15. Problem determination” on page 87.

Chapter 5. Configuring CICS Web support

This chapter explains how to set up CICS Web support. You must:

1. Ensure that you have the correct prerequisites. See “Prerequisites for using CICS Web support”
2. Specify the appropriate system initialization (SIT) parameters. See “System initialization parameters”
3. Create the necessary resource definitions. See “Defining resources to CICS” on page 32

Prerequisites for using CICS Web support

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

OS/390

The following must be installed on the OS/390 system:

- OS/390 eNetwork Communications Server Version 3.2.0 or above. Ports belonging to OS/390 eNetwork Communications Server 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.

The CICS region user ID must have an OS/390 UNIX System Services segment if it is to use CICS Web support.

CICS

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

Note: OS/390 eNetwork Communications Server CICS Sockets is not a prerequisite for CICS Web support.

OS/390 eNetwork Communications Server

Ports belonging to OS/390 eNetwork Communications Server must be made available for use by the CICS region involved.

New port numbers below 1024 must be defined to UNIX System Services.

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 YES, meaning that HTTP and IIOP services can process work. If TCPIP is set to NO, these services cannot be established, and you will not be able to use any TCPIPSERVICE resources defined with CEDA.
- You are recommended to migrate any existing TST macros to RDO and specify TST=NO in the system initialization table. If you have an assembled temporary storage table (TST) that does not specify MIGRATE=YES and that has not been migrated to RDO, message DFHAM4895 is issued during CICS initialization. This means that the installation of the default TSMODEL has failed, and CICS Web support will use auxiliary temporary storage.
- If you intend to use secure sockets layer (SSL), you must use:
  - the ENCRYPTION parameter to specify the level of encryption you want to use for TCP/IP connections using the SSL.
  - the KEYFILE parameter to specify the key database.

See the *CICS System Definition Guide* for details of system initialization parameters.

Defining resources to CICS

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
- The CICS Web support transient data queue for messages, CWBO
- A default TS queue model definition for DFHWEB. Note that because this is a model definition, it is subject to the rules governing the use of TS models in general. See the *CICS Transaction Server for OS/390 Release Guide* for details. If this definition fails to install because of a non-migrated TST that does not specify MIGRATE=YES, CICS will use auxiliary temporary storage.

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

The group DFH$WBSN contains the resource definitions for the security sample programs described in “Sample programs for security” on page 84.

Sample CICS Web TCPIPSERVICE 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.

Other changes that you might need to make are described in the next few sections.

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 template can reside in any of the

following places, and the data within it will be retrieved whenever a call is made for the template by means of an EXEC CICS DOCUMENT CREATE or EXEC CICS DOCUMENT INSERT command:

- MVS partitioned data set.
- CICS auxiliary temporary storage.
- CICS extrapartition transient data.
- CICS load module.
- CICS file.
- Exit program.

See the *CICS Resource Definition Guide* for details of how to define a DOCTEMPLATE. The *CICS Application Programming Guide* provides information about programming with documents and the associated EXEC CICS DOCUMENT commands.

MVS partitioned data set

You can use ISPF to create the templates as members of this data set. The record format can be FB (fixed blocked), VB (variable blocked), or U (undefined). The templates can contain sequence numbers as follows:

- VB format: the sequence numbers must be in record positions 1 through 8.
- FB 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 PDS, so members that are only partially sequenced might be interpreted incorrectly. The data set must be defined in a DD statement in the CICS JCL. The default DD name is DFHHTML. Multiple data sets can be concatenated on this statement.

Whenever you change the contents of a template in a PDS, you must re-install its associated DOCTEMPLATE definition; this lets CICS know that it must load a new copy of the template.

To allocate a PDS containing templates to a specific DD name in order to install templates from it, you can use the ADYN sample transaction. First install the DFH$UTIL group, which contains ADYN and its related programs, then run ADYN:

```
ADYN
ALLOC DDNAME(ddname) DATASET('template-pds') STATUS(SHR)
```

where *ddname* is the DDname specified in the DOCTEMPLATE definition, and *template-pds* is the name of the PDS containing the template to be installed. For further information on installing ADYN, see the *CICS Customization Guide*.

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 services that can be defined are CICS Web support and IIOP.

The TCPIPSERVICE definition allows you to manage these internal CICS interfaces, with CICS listening on multiple ports, with different flavors of CICS Web or IIOP 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 or HTTPSSL 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 OS/390 TCP/IP 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. If none is specified, CICS listens on all addresses used by OS/390 TCP/IP in the OS/390 region on which CICS is running.
- 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. This prefix should correspond to the prefix defined by an installed TS model definition. 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 TCPIPSERVICEs, see the *CICS Resource Definition 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 TCPIPSERVICE definition, it is started by the sockets listener task CSOL when a new connection request is received on the port specified on the TCPIPSERVICE 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(BELOW)  TASKDATAKEY(USER)
       RUNAWAY(SYSTEM)     SHUTDOWN(ENABLED)
       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.

TDQUEUE definitions

You must supply a definition for the CICS Web support message transient data queue. The following definition is provided in CICS-supplied group DFHWEB:

```
DEFINE TDQUEUE(CWBO)     GROUP(DFHWEB)
TYPE(INDIRECT)           INDIRECTNAME(CSSL)
DESCRIPTION(CICS Web interface message TD queue)
```

Setting up a PDS for the template manager

If you use the HTML template manager for constructing HTTP responses, you may provide an MVS partitioned data set to hold the templates. You can use ISPF to create the templates as members of this data set. The record format can be FB (fixed blocked), VB (variable blocked), or U (undefined). The templates can contain sequence numbers as follows:

- VB format: the sequence numbers must be in record positions 1 through 8.
- FB 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 PDS, so members that are only partially sequenced might be interpreted incorrectly.

Any DDname can be used to specify PDS 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 in DFHHTML. Multiple data sets can be concatenated on the DDname statement.

Defining a conversion table

If you have commarea-style Web application which do not use the Web API, 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=437,                                              *
      SRVERCP=037,                                              *
      RNAME=DFHWBHH,                                            *
      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 DFHWBHH. 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=437,                                              *
      SRVERCP=037,                                              *
      RNAME=DFHWBUD,                                            *
      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 DFHWBUD. 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.

```
DFHCNV TYPE=INITIAL
DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=DFHWBHH,USREXIT=NO,          *
      SRVERCP=037,CLINTCP=437
DFHCNV TYPE=SELECT,OPTION=DEFAULT
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,   *
      LAST=YES
DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=DFHWBUD,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
```

A sample DFHCNV table, DFHCNVSW, is provided.

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.

The OS/390 eNetwork Communications Server

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 MVS, new port numbers below 1024 must be defined to UNIX System Services.

To reserve the port on which CICS Web support listens for incoming client requests, you can specify the PORT option or the CICS jobname in the tcpip.PROFILE.TCPIP data set, as described in the *OS/390 eNetwork Communications Server: IP Configuration Guide*.

The maximum length of any queue of requests for a TCP/IP port on which a program is listening is controlled by the SOMAXCONN parameter in the tcpip.PROFILE.TCPIP data set. CICS listens on a TCP/IP port, so you must coordinate the value of this parameter with the value chosen for the Backlog parameter in the TCPIPSERVICE definition.

If you want full CICS function (that is, if you want to use DFH$WBSN and DFHWBENV), CICS Web support needs to access a name server during its operation. If the default name server is not suitable, you can specify another one by providing its address in a file allocated to the SYSTCPD DD statement in your CICS JCL (this sets the RESOLVER_CONFIG environment variable to the MVS dataset you have specified). The contents of this file are described in the *OS/390 eNetwork Communications Server: IP Configuration Guide*. You must specify at least the following:

NSINTERADDR *n.n.n.n*

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

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.

Chapter 6. The CICS WebServer Plugin

This supplied plugin enables a passthrough mechanism from the IBM WebSphere Application Server for OS/390 through the EXCI and into CICS Web support, using the CICS business logic interface.

Configuring the IBM WebSphere Application Server for OS/390

You have to change the configuration information in the IBM WebSphere Application Server for OS/390 if it is to use the CICS business logic interface to provide its service. *Webmaster's Guide Version 2 Release 1* gives details of the configuration statements.

You can use the following procedure:

1. You must set up CICS as follows:
   - Initialize the CICS region with ISC=YES
   - Install the RDO group DFHWEB.
   - Define a generic connection for EXCI (for example, by installing the sample group DFH$EXCI).
   - Ensure that IRC is open.
2. Define the CICSTS13.CICS.DFHDLL1 load library and CICSTS13.CICS.DFHEXCI to RACF® Program Control. RACF Program control notes the volume serial number of the volume containing the library, and does not allow the use of a different volume. If you later move the load library or the CICSTS13.CICS.DFHEXCI library to another volume, you must redefine it to RACF Program Control.
3. Add the CICSTS13.CICS.DFHDLL1 data set and the CICSTS13.CICS.DFHEXCI library to the STEPLIB concatenation in the JCL for the IBM WebSphere Application Server for OS/390.
4. Use the following command in the directory that contains the httpd.conf file for the IBM WebSphere Application Server for OS/390:

   ```
   ln -e DFHWBAPI dfhwbapi.so
   ```

   When it is used in the STEPLIB concatenation, this command establishes a link from the IBM WebSphere Application Server for OS/390's home directory to the DLL dfhwbapi.so in member DFHWBAPI in the CICSTS13.CICS.DFHDLL1 library.
5. Add one or more directives of the following format to the httpd.conf file:

   ```
   Service  /sourceurl/*  /home/dfhwbapi.so:DFHService/targeturl/*
   ```

   where the values are:

   home is the directory that contains the httpd.conf file for the IBM WebSphere Application Server for OS/390.

   sourceurl
   is a string of characters that selects an incoming URL to be processed by DFHWBAPI. The asterisk following it is a wildcard string representing the remaining characters of the incoming URL. *sourceurl* can be in any format, so details such as the *applid* and the *transaction* can be hidden from end users.

targeturl

targeturl is a string of characters that will be analyzed as the URL by DFHWBAPI after the wildcard characters are appended. The target URL must contain the applid of the target CICS region as its first subfield, which must be followed by the standard fields for a CICS URL, as described in “The default analyzer” on page 46. This means that the target URL, after substitution of the wildcard, must be in the format:

```
/applid/converter/tran/program/filename
```

where the values are:

applid the application id of the target CICS region

converter
the name of the converter program to be used in the CICS region, or CICS if no converter is to be used.

tran the transaction to be executed in the CICS region. Because the transaction is the target of an EXCI request, it should not be the Web alias transaction CWBA, but should be a mirror transaction, such as CSM3. The transaction receives *targeturl/**, not *sourceurl/**, as the incoming URL.

program
the name of the program to be executed in the CICS region.

filename
is any further information that will be examined by *program*.

If *targeturl* is omitted, the incoming URL is passed directly to DFHWBAPI and must therefore be in the format just described. You can use the mapping from *sourceurl* to *targeturl* to change the URL format from the standard one expected by CICS into the format expected by DFHWBAPI. Use the following directive to do this:

```
Service  /cics/cwba/*  /home/dfhwbapi.so:DFHService/applid/CICS/CSM3/*
```

6. Some of the CICS-supplied template definitions for CICS-supplied transactions contain references to graphics files in the format:

   ```
   /dfhwbimg/filename
   ```

   where DFHWBIMG is a special-purpose CICS-supplied converter program used by the CICS Web bridge. If you want such graphics files to be displayed correctly, you should include a directive as follows:

   ```
   Service  /dfhwbimg/*  /home/dfhwbapi.so:DFHService/applid/DFHWBIMG/CSM3/*
   ```

   where applid specifies the CICS system that will supply the graphics files (this may not be the same CICS system that does the bridge work).

Chapter 7. Writing an analyzer for CICS Web support

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

The analyzer

There is an analyzer for each CICS Web support TCPIPSERVICE. You can supply your own analyzer, or you can use the CICS-supplied analyzer, DFHWBADX. The place of the analyzer in CICS Web support is illustrated in Figure 9 on page 22. The analyzer is expected to use information in the incoming request to decide what CICS resources are needed to process the request. It 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 205.

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 205.

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:

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

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

If the analyzer is for a connection specified as SSL(CLIENTAUTH) in the TCPIPSERVICE definition, a userid derived from the client certificate may be provided as an input.

If the request is not an HTTP request, 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 DFHWBTTA.
- 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 TCPIPSERVICE, allowing the port number of the TCPIPSERVICE 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 45.

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

If the request can be 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 will be 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 92.

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 205. 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 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 be using. 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.

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 J. HTML coded character sets” on page 257. 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.

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, particularly where SSL is being used. To enable CICS persistent connections support you must specify either NO or a numeric value for the SOCKETCLOSE keyword on the relevant TCPIPSERVICE 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, rather than in an MVS PDS. 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 1 and 8 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 1 and 4 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 1 and 8 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 10. 3270 applications on the Web” on page 55 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 209.

Chapter 8. Writing a converter

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

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 forst 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 211.

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 9 on page 22 and Figure 10 on page 24. The converter must run in the same CICS region as the TCPIPSERVICE 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.

Writing a converter—general

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.

Inputs

The converter input includes:

- An indicator of the function (**Decode** or **Encode**) that is to be performed
- Parameters for the function, as described in later sections

Outputs

The converter output must include a response, and might include a reason code. The outputs are described in more detail for each function.

Processing

The inputs and outputs are presented in a CICS communication area. On entry to the converter, it should check the input field **converter_function** to see whether the requested function is **Decode** or **Encode**. The rest of the processing depends on the function requested.

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

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 buffer can occupy the same storage as the communication area returned by the CICS program, or you can use EXEC CICS GETMAIN to get new storage. After using EXEC CICS GETMAIN in this way, you must set the output pointer to the new storage, then use EXEC CICS FREEMAIN to free the storage when you have finished with it.
- 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 211. 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.

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**

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 221. 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 buffer can occupy the same storage as the communication area returned by the CICS program, or you can use EXEC CICS GETMAIN to get new 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 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 211. 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.

Chapter 9. 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 TCPIPSERVICE. 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 235. For more information on user-replaceable modules, see the *CICS Customization Guide*.

Chapter 10. 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:

- 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/dfhwbtta/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.
- Continuation requests are requests that continue a conversation or pseudoconversation. DHFWBTTA 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 two 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 1 and 8 characters long. It is ignored by DFHWBTTA.
- *alias* must be between 1 and 4 characters long. It is ignored by DFHWBTTA.
- *program* must be between 1 and 8 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 will be 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 will be 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 55. 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 11. Creating HTML templates from BMS definitions” on page 61.

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 60). 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/1.3.0">
</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.
- 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 55, 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="PF1">
<input type="submit" name="DFH_PF2"  value="PF2">
<input type="submit" name="DFH_PF3"  value="PF3">
<input type="submit" name="DFH_PF4"  value="PF4">
<input type="submit" name="DFH_PF5"  value="PF5">
<input type="submit" name="DFH_PF6"  value="PF6">
<input type="submit" name="DFH_PF7"  value="PF7">
<input type="submit" name="DFH_PF8"  value="PF8">
<input type="submit" name="DFH_PF9"  value="PF9">
<input type="submit" name="DFH_PF10"  value="PF10">
<input type="submit" name="DFH_PF11"  value="PF11">
<input type="submit" name="DFH_PF12"  value="PF12">
<br>
```

```
<input type="submit" name="DFH_PF13"  value="PF13">
<input type="submit" name="DFH_PF14"  value="PF14">
<input type="submit" name="DFH_PF15"  value="PF15">
<input type="submit" name="DFH_PF16"  value="PF16">
<input type="submit" name="DFH_PF17"  value="PF17">
<input type="submit" name="DFH_PF18"  value="PF18">
<input type="submit" name="DFH_PF19"  value="PF19">
<input type="submit" name="DFH_PF20"  value="PF20">
<input type="submit" name="DFH_PF21"  value="PF21">
<input type="submit" name="DFH_PF22"  value="PF22">
<input type="submit" name="DFH_PF23"  value="PF23">
<input type="submit" name="DFH_PF24"  value="PF24">
<br>
<input type="submit" name="DFH_PA1"   value="PA1">
<input type="submit" name="DFH_PA2"   value="PA2">
<input type="submit" name="DFH_PA3"   value="PA3">
<input type="submit" name="DFH_CLEAR" value="Clear">
<input type="submit" name="DFH_ENTER" value="Enter">
<input type="submit" name="DFH_PEN"   value="Pen">
<input type="reset"  value="Reset">
</form>
</body>
</html>
```

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.

Chapter 11. 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 limitiations, be recovered using DFHBMSUP. See *CICS Operations and Utilities Guide* for details.

Standard generation

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 PF1 to PF24
  - HTML reset
- Up to five hidden variables, DFH_NEXTTRANSID.1 to DFH_NEXTTRANSID.5, whose values are the names of the first five fields in the map. The use of these variables is explained in “Chapter 10. 3270 applications on the Web” on page 55.
- 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 63.

Customization facilities

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

- The DFHMDX macro: 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 61.
- 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 61.

(For customization number 9 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 SYSLIB DD statement for the assembler. Write appropriate DFHWBOUT macro invocations, and put them in the appropriate places in your map definitions.
3. Assemble the existing map definitions with TYPE=TEMPLATE on the DFHMSD macro, or SYSPARM=TEMPLATE in the parameters passed to the assembler. Note that the label on the DFHMSD macro is used to name the HTML templates produced for each map in the 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 IEBUPDTE source statements that set up one template for each map in a mapset.
4. Use IEBUPDTE to store the templates in the template library. If the record format of the template library is not fixed blocked, you will need to store them in another PDS, and then convert them to the record format of the template library using, for instance, ISPF COPY.

5. If you want to put your templates in a PDS other than the one specified in the DFHHTML DDname, you must define DOCTEMPLATE definitions for your templates, and specify an alternate DDname. The alternate DDname must also be specified in your CICS JCL.

   To allocate a PDS containing templates to a specific DD name in order to install templates from it, you can use the ADYN sample transaction. First install the DFH$UTIL group, which contains ADYN and its related programs, then run ADYN:

   ```
   ADYN
   ALLOC DDNAME(ddname) DATASET('template-pds') STATUS(SHR)
   ```

   where *ddname* is the DDname specified in the DOCTEMPLATE definition, and *template-pds* is the name of the PDS containing the template to be installed. For further information on installing ADYN, see the *CICS Customization Guide*.

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 first invocation will usually set defaults for the values to be applied to subsequent invocations of DFHMDX by specifying * for the mapset name and map name. Later invocations will override or add to the parameters for specific maps in the mapset.
4. A MEND statement to end the definition.

Customization examples

The following sample shows a customizing macro definition. The continuation characters are in column 72, and the continued text is resumed in column 16.

```
MACRO
DFHMSX
DFHMDX MAPSET=*,MAP=*,PF3='Exit'
DFHMDX MAPSET=DFHWB0,MAP=DFHWB01,                              *
      TITLE='CICS Web Interface',                              *
      PF1='Help',PF9='Messages'
DFHMDX MAPSET=DFHWB0,MAP=DFHWB02,                              *
      TITLE='CICS Web Interface Enable',                       *
      PF1='Help',PF4='Save',PF9='Messages',PF12='Return'
MEND
```

In this example, the first DFHMDX macro specifies a default value for the PF3 keyword of any subsequent occurrence of DFHMDX. The value applies to every subsequent DFHMDX macro irrespective of its mapset name or map name unless it contains the PF3 keyword. The other two DFHMDX macros specify particular treatment for two maps in the DFHWB0 mapset. If the assembler input contains any other mapsets or maps than these, they will all be converted as if they had DFHMDX macros with matching mapset name and map name and PF3=’Exit’.

The customizations listed in “Why customize the generation of templates?” on page 61 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 PF1.

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="&WBDATE&W*
           BTIME GMT">'
    ```

    will produce the following lines in the head section of the HTML template:

    ```
    <meta name="author" content="E Phillips Oppenheim">
    <meta name="owner"  content="epoppenh@xxxxxxx.yyy.com">
    <meta name="review" content="19980101">
    <meta http-equiv="Last-Modified" content="23-Dec-1997 12:06:46 GMT">
    ```

    DFHMSD sets the values of &WBDATE and &WBTIME to the time and date at which the macro is assembled.

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.
- You should not mix BMS commands and terminal control commands, as this can cause transaction to end abnormally.
- 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.

The DFHMDX macro

Figure 12 on page 67 shows the syntax of the DFHMDX macro.

DFHMDX

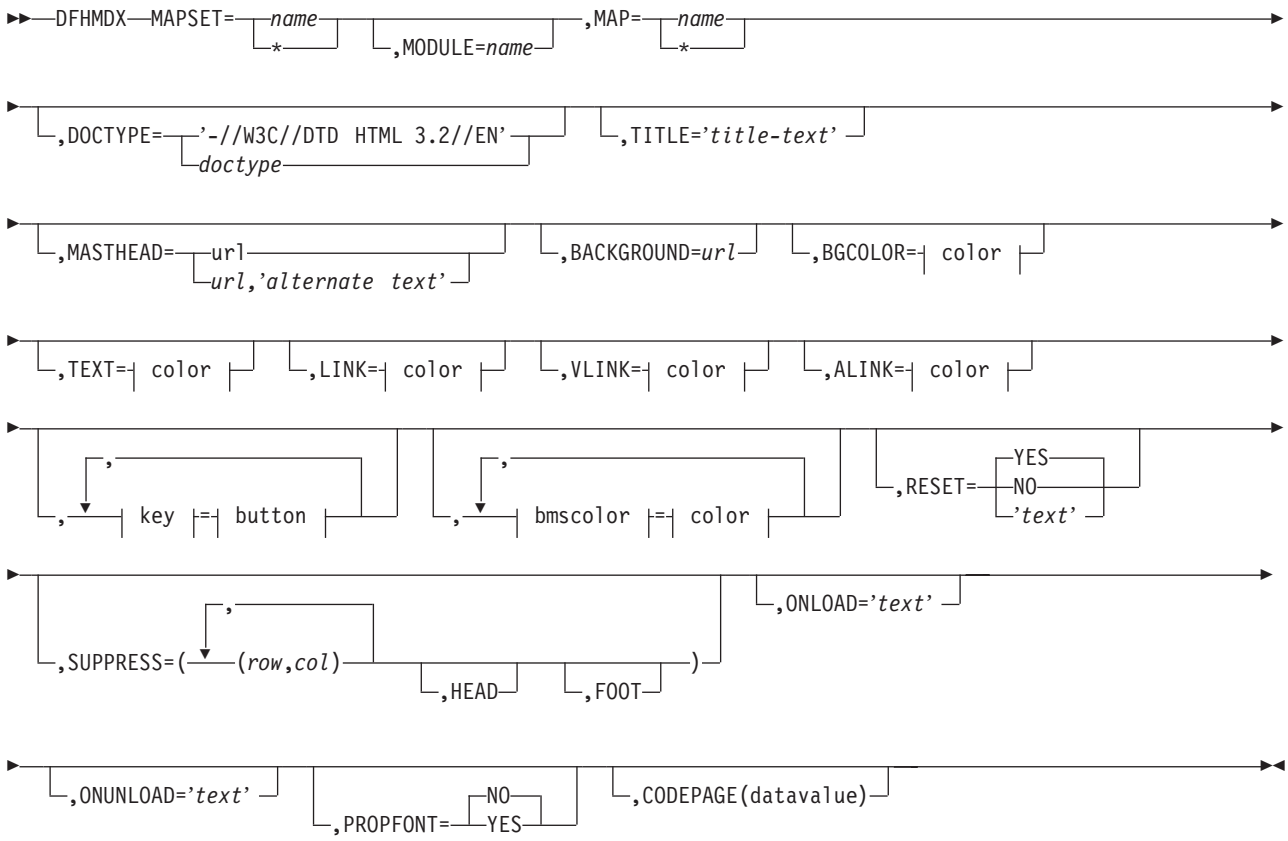


Figure 12. Syntax of DFHMDX

The keyword parameters to this macro can appear in any order.

MAPSET
specifies the name of the mapset that contains the map to which other options refer. If you specify an asterisk, the options become the default to all subsequent mapsets.

MODULE
specifies the name of the load module into which the mapset is link-edited. You can only use this parameter if you do not specify MAPSET=*. The name you specify (which can only be seven characters) is used to construct the names of the templates by adding a single character suffix. The default value is the name of the 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.

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 PF1 to PF24, PA1 to PA3, CLEAR, ENTER, and PEN.

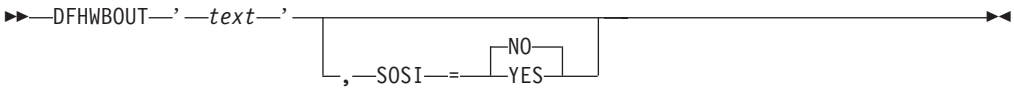
button can be (IMAGE,*url*), where *url* specifies the URL of a graphic image to be used for the button, or '*text*', where *text* is the text to be put in the button, or NO if the button is not to appear.

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

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 12. 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.

- "HTTP requests" describes HTTP requests and how CICS handles them.
- "HTTP responses" on page 73 describes HTTP responses and how a CICS program can construct them.
- "Sample application programs" on page 77 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.

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 xxi.

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 the OS/390 eNetwork Communications Server, 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 7. Writing an analyzer for CICS Web support” on page 43. The default analyzer, the URLs that it accepts, and the way it interprets them, are described in “The default analyzer” on page 46.

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:

- Use the EXEC CICS WEB commands (this is the recommended method). See “Using EXEC CICS WEB commands to receive an HTTP request”.
- Use the environment variables program DFHWBENV (this method is retained for compatibility with previous releases). See “Using DFHWBENV to retrieve information from an HTTP request” on page 73.

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), and the protocol version in use.

The WEB READ/STARTBROWSE/READNEXT/ENDBROWSE commands allow the application to extract header information that it wants to read from the 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 and the content length.

The EXEC CICS WEB RECEIVE command allows the application to receive the message body of the HTTP request into a buffer. Refer to `http://w3.org/protocols/rfc1945/rfc1945` for a full specification.

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 227 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.

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 TCPIP 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 retained for compatibility with earlier releases.

How to send an HTTP response

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

- Use the EXEC CICS application programming interface (this is the recommended method).
- Use the HTML template manager (this method is retained for compatibility with previous releases).

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 Application Programming 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 221 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.

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 TCPIPSERVICE, the way in which CICS handles escaped data depends up the analyzer being used for that TCPIPSERVICE.

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. For example, if you have an application which is run both by the WebServer Plugin for IBM WebSphere Application Server for OS/390, and a CICS Web TCPIPSERVICE, you should set WBRA_UNESCAPE to WBRA_UNESCAPE_NOT_REQUIRED, to ensure that the body of the HTTP request passed to the application is in the same form, irrespective of the caller of the Web application.

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 retained for compatibility with earlier releases.

The template manager maintains a symbol table for each active page environment. In WBTL_BUILD_HTML_PAGE and WBTL_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 ampersands as separators. 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 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`.

Using the output of the environment variables program

The environment variables program, DFHWBENV, is retained for compatibility with earlier releases.

The output of the environment variables program, described in "Appendix E. Reference information for DFHWBENV" on page 227, 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 81. 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 13. 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
 Certificate   ==>
 STatus        ==> Open
 SSl           ==> No
 TRansaction   ==> CWXN
 Backlog       ==> 00001
 TSqprefix     ==>
 Ipaddress     ==>
```

2. Define and install two DOCTEMPLATE definitions. One specifies a MEMBERNAME of TEMPLATE1 and a TEMPLATENAME=WEBDISPLAY1, as follows:

```
 DOctemplate  ==> MYDOCT
 Group        ==> MYGROUP
 DEscription  ==> Template for display on Web
FULL TEMPLATE NAME
 TEmplatename ==> WEBDISPLAY1
ASSOCIATED CICS RESOURCE
 File         ==>
 TSqueue      ==>
 TDqueue      ==>
 Program      ==>
 Exitpgm      ==>
PARTITIONED DATA SET
 DDname       ==> DFHHTML
 Membername   ==> TEMPLATE1
```

   The second specifies a MEMBERNAME of TEMPLATE2 and a TEMPLATENAME of WEBDISPLAY2.

3. Create the first template. In this example, the HTML data is in an MVS PDS accessed in the JCL by DD statement DFHHTML in member TEMPLATE1:

```
<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://mytso: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 81.

4. Create the second template. In this example, the data is HTML in an MVS PDS accessed in the JCL by DD card DFHHTML in member TEMPLATE2:

```
<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 J. HTML coded character sets” on page 257.

6. In your PROGRAM2 application program, define a 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(DOCLENGTH)
     MAXLENGTH(80)
     CLNTCODEPAGE('client codepage')
     HOSTCODEPAGE('host codepage')
```

where CLNTCODEPAGE can be any of the client codepages listed in "Appendix J. HTML coded character sets" on page 257, 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(DOCLENGTH)
```

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 J. HTML coded character sets" on page 257.

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 7. Writing an analyzer for CICS Web support" on page 43). The format is:

http is the protocol you want the browser to use. This can be **http** or **https**. HTTPS is a variant of HTTP, used for handling secure transactions. You should use the HTTPS protocol only if you have specified SSL=YES or SLL=CLIENTAUTH in the TCPIPSERVICE definition (see "Part 4. Using secure sockets layer (SSL)" on page 105 for information about SSL).

yoursystem
is the TCP/IP name or IP address for the OS/390 system on which CICS is running. If none is specified, this defaults to the IP address of the default TCP/IP stack for the OS/390 region 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. If you are using the HTTPS protocol and you omit the port number, it defaults to 443.

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 14. Security for CICS Web support

This chapter is organized as follows:

- "Security for the CICS Web support" describes security considerations for the HTML template manager PDS, and the CICS Web support transactions.
- "Sample programs for security" on page 84 describes the operation of the sample security analyzer, converter, and sign-on program.

Security for the CICS Web support

This section describes security considerations for the HTML template manager PDS, and the CICS Web support transactions.

Security for the HTML template manager PDS

If your CICS programs use the partitioned data set facilities of the HTML template manager described in "Appendix D. Reference information for DFHWBTL" on page 221, the CICS region user ID must have READ authority for the data set described in the DOCTEMPLATE PDS definition. If you reference other partitioned data sets by defining DOCTEMPLATEs with other DDnames, 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 RACF 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 46, 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 are running with SSL=CLIENTAUTH (either as a SIT parameter or on a

TCPIPSERVICE definition), **wbra_userid** *may* contain a user ID on input to the analyzer. 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

Sample programs for security

Two sets of sample programs are provided:

- 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$WBSN
- The basic authentication sample programs, described in "The basic authentication sample programs" on page 85:
  - The basic authentication analyzer, DFH$WBAU
  - The basic authentication converter, DFH$WBSB

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

The security sample programs

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.

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. The Web browser receives the redirected URL, and sends a request for the original program with the token that identifies the validated user ID.
11. 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.

The 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 15. 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 *CICS Messages and Codes*

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 *CICS Messages and Codes*.

This chapter is organized as follows:

- “Recovery procedures (CICS Web support)” on page 88 describes how CICS Web support copes with software errors.
- “Product design considerations (CICS Web support)” on page 88 describes aspects of the design of CICS Web support that you need to know for problem determination.
- “Troubleshooting” on page 88 describes a method of analyzing problems in CICS Web support and CICS business logic interface.
- “Using messages and codes” on page 89 describes how to find information about messages and abend codes.
- “CICS Web support and CICS business logic interface trace information” on page 90 describes CICS Web support and CICS business logic interface trace information.
- “Dump and trace formatting” on page 91 describes how to control the formatting of dumps and trace entries.
- “Debugging the user-replaceable programs” on page 91 gives hints about debugging user-replaceable programs.

Recovery procedures (CICS Web support)

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

If OS/390 eNetwork Communications Server 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 TS release
   - OS/390 release
   - 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 OS/390 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 TS
   - To the OS/390 eNetwork Communications Server
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 12 on page 67.)
8. What was the state of the OS/390 eNetwork Communications Server? (Try the **netstat** command.)

Documentation about the problem

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

- System dump: This contains the CICS internal trace
- CICS auxiliary trace, if enabled
- TCP/IP trace
- GTF 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 DFH*xxnnnn*., 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, and Korean. The CICS message editing utility can be used to translate them into other languages supported by CICS, as described in the *CICS Operations and Utilities Guide*.

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)

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 VERBEXIT in the IPCS control statement for dump formatting as follows:

```
IPCS VERBEXIT CICS530 WB=0|1|2|3,TR=1|2
```

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.

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 IPCS output by searching for `===WB`. It is under the heading `CICS Web support`.

The CICS Sockets output can be found in the IPCS output by searching for `===SO`. It is under the heading `CICS Sockets`.

The document domain output can be found in the IPCS 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)
- 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.

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.

It contains:

Chapter 16. Introduction to 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 19, 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 12.

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

“Chapter 17. Configuring the CICS business logic interface” on page 103 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):
  - Web browsers connected to the IBM WebSphere Application Server for OS/390. In the IBM WebSphere Application Server for OS/390, a user-provided Common Gateway Interface (CGI) script builds the EXCI request to the CICS business logic interface.
  - Java applications using CICS-provided Java classes and local Gateway facilities. The applications can create `ECIRequest` objects that communicate with CICS without using a CICS Transaction Gateway. See “Part 5. Using the CICS Transaction Gateway for OS/390” on page 115 for more information about the CICS-provided Java classes and the local Gateway facilities.
- 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.
- ONC RPC clients. These programs can use CICS ONC RPC support to call the CICS business logic interface.

Processing examples

Figure 13 shows how the CICS business logic interface processes a request from an MVS application that uses the EXCI.

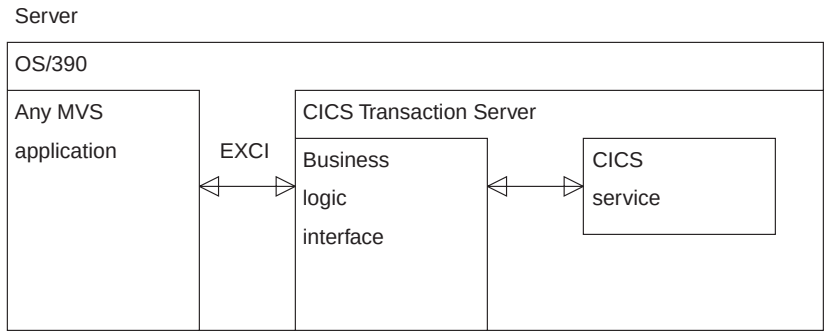


Figure 13. Processing a request from the EXCI

The MVS 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 TS provides the requested service, and returns any output in the communication area.

Figure 14 shows how the CICS business logic interface processes a request from a CICS client that uses the ECI.

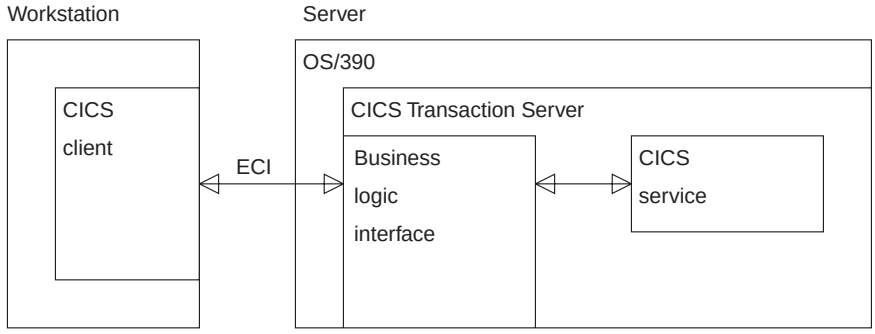


Figure 14. Processing a request from the ECI

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 TS provides the requested service, and returns any output in the communication area. The ECI operates with either the SNA protocol or with TCP62, which allows a SNA connection over TCP/IP (see the *CICS Family: Client/Server Programming* for further information)..

Control flow in request processing

To make decisions about the facilities you will use, and how you will customize them, you need to understand how the components of the CICS business logic interface interact.

Using the CICS business logic interface to call a program

Figure 15 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.

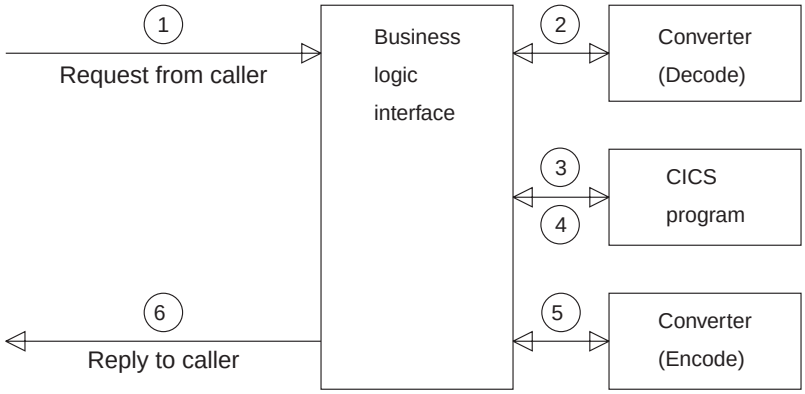


Figure 15. Calling a program with the CICS business logic interface—control flow

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 16 on page 98 shows the control flow through the CICS business logic interface for a request for a terminal-oriented transaction. Note that the business logic interface is running under a CICS mirror transaction, not a Web CICS transaction. The first part of the processing is the same as for calling a program, but if you want to run a transaction, you must specify DFHWBTTA as the CICS program to be called, in **wbbl_server_program_name**.

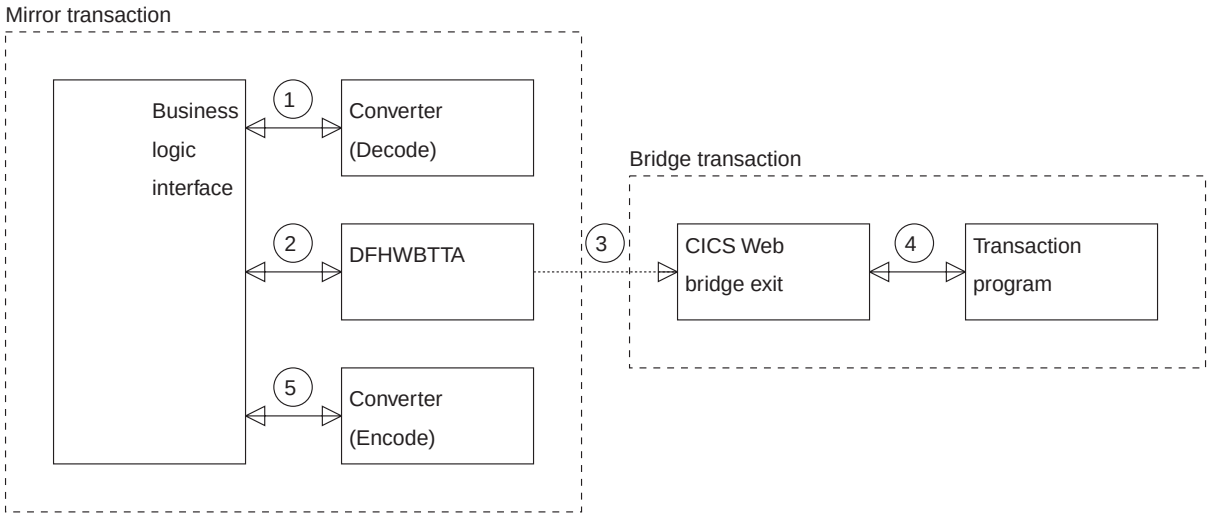


Figure 16. Running a transaction with the CICS business logic interface—control flow

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 DFHWBTTA.
2. The CICS business logic interface 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.
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.

Data flow in request processing

To make decisions about the facilities you will use, and how you will customize them, you need to understand how data is passed in the CICS business logic interface.

Using the CICS business logic interface to call a program

Figure 17 on page 99 shows the data flow through the CICS business logic interface to a program, and back to the requester.

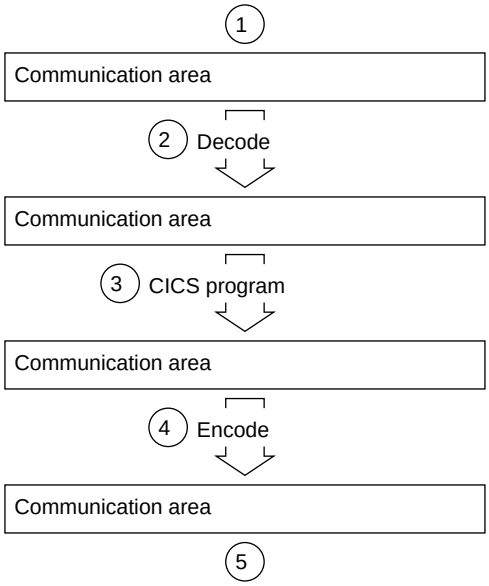


Figure 17. Calling a program with the CICS business logic interface—data flow

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.

Request for a terminal-oriented transaction

Figure 18 on page 100 shows the data flow for a request that starts a terminal-oriented transaction.

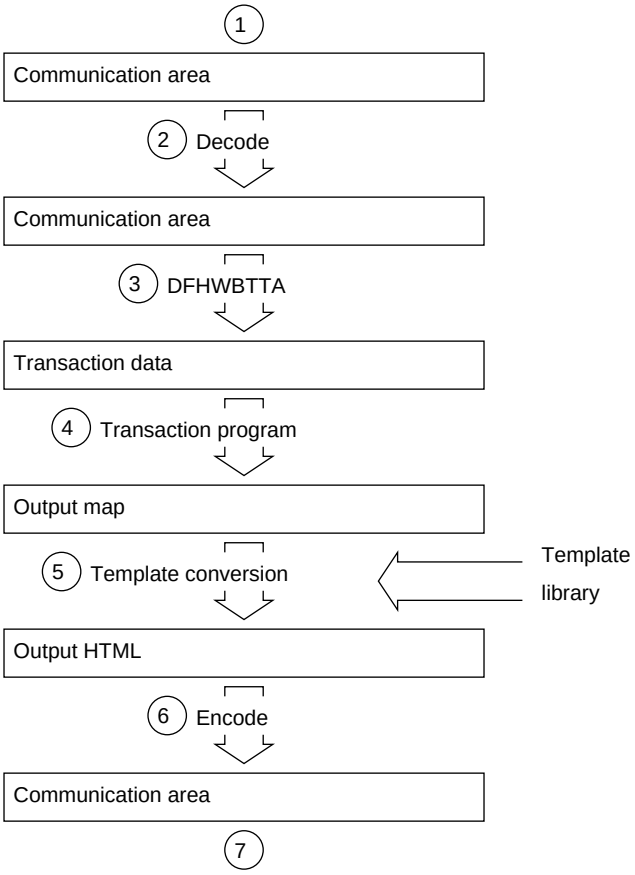


Figure 18. Starting a terminal-oriented transaction—data flow

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 11 on page 25.

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

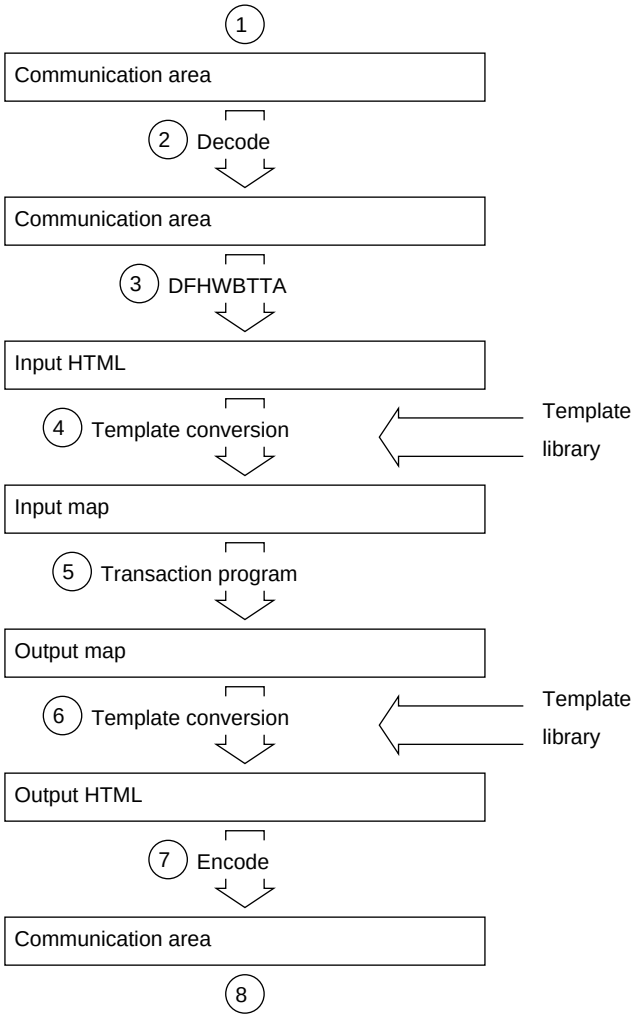


Figure 19. Continuing a terminal-oriented transaction—data flow

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 11 on page 25.

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 17. 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 15 on page 97 and Figure 16 on page 98. 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:

- Pointer mode
- Offset mode

The differences in the functions are described in “Chapter 8. Writing a converter” on page 49, and in “Appendix C. Reference information for the converter” on page 211. Converters called in offset mode are more restricted than converters called in pointer mode. All requests from any of the following sources result in offset mode calls to the CICS business logic interface:

- Web browsers using the IBM WebSphere Application Server for OS/390.
- Java applications using the local gateway function.
- DCE RPC clients.
- Web browsers using the CICS Transaction Gateway or the CICS Transaction Gateway for OS/390.

You must set the WEBDELAY system initialization parameter, as described in “System initialization parameters” on page 31.

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 197

Part 4. Using secure sockets layer (SSL)

This part of the book describes secure sockets layer (SSL). It contains:

Chapter 18. Introduction to secure sockets layer (SSL)

This chapter provides an overview of how secure sockets layer (SSL) provides transaction security for CICS communications over TCP/IP. It includes the following sections:

Overview of SSL

SSL is a **handshake protocol** developed to provide security and privacy over the Internet. The SSL protocol uses encryption and authentication to ensure:

Privacy

The data to be exchanged between the client and the server is encrypted, so that only that client and that server can make sense of the data. SSL uses public key encryption as a secure mechanism to distribute a secret key between the server and the client. Public key encryption is a technique that uses a pair of asymmetric keys for encryption and decryption. With SSL, a secret (symmetric) key is passed between the client and the server, using public key cryptography, and the key is then used to encrypt and decrypt all traffic along the SSL connection. This encryption protects the data from other parties trying to eavesdrop, as no other parties will have the secret key needed to decrypt the data. This ensures that private information such as a credit card number is transferred securely.

Integrity

The message transport includes a message integrity check based on a secure hashing algorithm. This algorithm is performed when the message is sent, and again when it is received. If the two hash values do not match, the receiver is warned that the message may have been tampered with. In the US, a 128–bit encryption key is used, in France a 40–bit encryption key is used, and in the rest of the world a 56–bit encryption key is used.

Authentication

When a client establishes a connection with CICS, it is required to authenticate its details to the server. The authentication mechanism is based on the exchange of digital certificates (X.509v3 certificates). These digital certificates contain information about an entity, such as the system name and public key, and the server's digital signature. Digital certificates are issued by a Certificate Authority (CA), and encrypted using the CA's private key. If you can decrypt the certificate using the CA's public key, you know that the information contained within the certificate can be trusted (that is, that the certificate really does belong to whoever claims to own it).

SSL and the Web

The HTTPS protocol is a variant of HTTP for handling secure Web transactions. Most current browsers support the HTTPS URL access method to connect to HTTP servers that use SSL. A secure connection is made with a URL such as **https://www.company.com**

If you use the HTTPS protocol without specifying a port number, a default port number of 443 is assumed.

Encryption and keys

The SSL protocol operates between the application layer and the TCP/IP layer. This allows it to encrypt the data stream itself, which can then be transmitted securely, using any of the application layer protocols. Two encryption technique are used:

- Public key cryptography standard (PKCS), which encrypts and decrypts certificates during the SSL handshake. Encryption keys are created in pairs, a public key and its associated private key. Data encrypted with a given public key can be decrypted only with the associated private key; this means that data is readable by only the intended recipient. Data encrypted with a given private key can be decrypted only with the associated public key; this means that authentication data is assured to originate from the owner of the private key.
- A mutually agreed symmetric encryption technique, such as DES (data encryption standard), or triple DES, is used in the data transfer following the handshake.

PKCS, as used by SSL, works briefly as follows:

1. A key-pair is requested, usually as part of certificate application (see “Authentication and certificates” on page 109.
2. As part of the certificate creation, a private key and public key are created by means of an algorithm based on two random numbers. The resultant two keys are related to each other, but it would be very difficult to deduce one from the other.
3. The private key is stored securely, and is not made known to anyone but its owner. The public key is freely available to anyone.

The private and public key pair is then used during the SSL handshake as follows:

1. A wants to send a private message to B, so A first encrypts the message using B’s public key, which is freely available.
2. On receiving the message, B decrypts it with B’s private key. The relationship between the public and private key is such that anything encrypted with a given public key can be decrypted only by the associated private key. As long as the private key is not divulged, any data encrypted with the associated public key is safe.

In CICS, the system’s private and public key pair are held in a “key database”, which is stored within the hierarchical file system (HFS) of OS/390 and which is managed by the **gskkyman** utility.

Authentication and certificates

To make an environment secure, you must be sure that any communication is with ″trusted″ sites whose identity you can be sure of. SSL uses certificates for authentication — these are digitally signed documents which bind the public key to the identity of the private key owner. Authentication happens at connection time, and is independent of the application or the application protocol.Authentication involves making sure that sites with which you communicate are who they claim to be. With SSL, authentication is performed by an exchange of certificates, which are blocks of data in a format described in ITU-T standard X.509. The X.509 certificates are digitally signed by an external authority known as a certificate authority. Using a search string such as **certificate authority**, search the Web for companies that provide this service.

Certificates are digitally signed using the public-key encryption technique. The signature is created by partially encrypting the certificate with the certificate authority's private key. A user of the certificate is assured of the origin of the certificate when it is successfully decrypted by the certificate authority's public key.

In SSL, the server certificate is mandatory, but the client certificate is optional, and it is up to the server (that is, CICS) to decide whether to accept a connection from a client without a certificate.

In CICS, the required server certificate and related information about certificate authorities are held in a "key database", which is stored within the hierarchical file system (HFS) of OS/390. This file contains your system's private and public key pair, together with your server certificate and the certificates for all the certificate authorities that might have signed the certificates you receive from your clients.

Chapter 19. Configuring CICS to use SSL

This chapter explains how to configure CICS to use SSL. It contains:

Hardware prerequisites

SSL does not require any additional hardware, but performance is improved if the appropriate cryptographic hardware is installed. The Cryptographic Coprocessor Feature is an optional feature of IBM S/390® Multiprise® 2000 and IBM S/390 Parallel Enterprise Server™ Generation 3 systems. It is standard on IBM S/390 Parallel Enterprise Server Generation 4 systems. The IBM 4753–014 Network Security Processor can be attached to S/390 systems that do not include the Cryptographic Coprocessor Feature.

Software prerequisites

These are the software prerequisites for using SSL:

- OS/390 Version 2 Release 7
- APAR PQ23421 must be applied to CICS TS Release 3 to enable SSL.

System set-up

The following tasks are necessary for SSL to work with CICS:

- Obtain an X.509 certificate from a certificate authority such as Verisign, whose Web page is at:

  `http://www.verisign.com/`
- Create a key database to hold your public and private keys, your server certificate, and the certificate information for each client you expect to communicate with. Use the **gskkyman** utility (supplied with OS/390 Unix System Services) to do this.

  When you create the key database with gskkyman, you will be prompted for a password. This password is used to protect the database, and you will be required to enter it whenever you access the database. Alternatively, you can use gskkyman to create a stashed password file, which allows CICS to access the database without specifying the password.

  When you add a server certificate to the key database, you can give the certificate a name, or certificate label. You can also choose to make one of the certificates the default certificate for that database.

- Ensure that the CICS region userid is an MVS OpenEdition® userid that is authorized to read the HFS file specified in the KEYFILE attribute.
- If you intend to use client certificates to select the userids under which CICS Web transactions execute, you need to receive the certificates from your clients and install them in your External Security Manager. If you are using the S/390 Security Server, use the RACDCERT TSO command to do this. To add a certificate, the command is:

  ```
  RACDCERT ADD(datasetname) ID(userid) TRUST
  ```

  For further information about the RACDCERT command, see the*CICS RACF Security Guide*.

System initialization parameters

There are three system initialization parameters relating to SSL:

- ENCRYPTION(WEAK|NORMAL|STRONG). Use this to specify the level of encryption to be used. STRONG specifies 128–bit encryption (available in the US only), WEAK specifies 40–bit encryption (used in France), and NORMAL specifies 56–bit encryption (available in all other countries).
- KEYFILE. Use this to specify the HFS pathname of the key database. If you did not create a stashed password file for the key database with gskkyman, you can specify the database password here by appending a backslash (\) to the pathname, followed by the password. For example:

  ```
  KEYFILE=/u/cicsssl/keys/key.kdb\password
  ```

- TCPIP(YES|NO). TCPIP specifies whether CICS TCPIP services are to be activated at CICS startup. The default is YES, meaning that HTTP and IIOP services can process work. If TCPIP is set to NO, these services cannot be enabled.

See the *CICS System Definition Guide* for details of these system initialization parameters.

Resource definitions

Install and activate a TCP/IP service for SSL use, either with or without client authentication. The TCPIPSERVICE resource definition has two attributes relating to SSL:

- PORTNUMBER. This is the TCP/IP port number on which the SSL service will be provided.
- CERTIFICATE(*certificate-label*). The TCP/IP service uses one of the certificates in the key database as its server certificate. Use the CERTIFICATE option to specify a particular certificate (*certificate_label*is the name that you assigned to the certificate with gskkyman). If you do not specify CERTIFICATE, CICS uses the default certificate in the key database.
- SSL(NO|YES|CLIENTAUTH). Use this to specify the level of SSL to be used. NO specifies no SSL support. YES specifies that SSL support is to be activated, and clients connecting to the specified port number must use the SSL protocol to connect with CICS (that is, they must specify ″https″ rather than ″http″ as the protocol in the URL used to access the service). CLIENTAUTH specifies that the client, as wel as the server, must have a certificate. The client certificate is received by CICS during the SSL handshake, and can be used either to

determine the userid under which the CICS transaction can be executed, or to provide information about the client by means of the EXEC CICS EXTRACT CERTIFICATE command.

The TCPIPSERVICE definition must be activated, either by specifying STATUS(OPEN) and installing the definition, or by installing the definition and later using a CEMT SET TCPIPSERVICE OPEN command. The TCPIPSERVICE definition is described in detail in the *CICS Resource Definition Guide*.

System programming

The SSL attribute of the INQUIRE TCPIPSERVICE command returns the level of secure sockets support being used for this service (SSL=NO, SSL=YES, or SSL=CLIAUTH).

Application programming

Examine existing application programs to see whether they can exploit the EXEC CICS EXTRACT CERTIFICATE command. This command allows you to extract information from any client certificate received over an SSL connection. See the *CICS Application Programming Reference* for details of the command.

A sample application program: DFH0WBCA

DFH0WBCA is a sample program provided by CICS. It demonstrates how you can extract information from an SSL client certificate and construct the response as a CICS document with the EXEC CICS DOCUMENT commands. The *CICS Application Programming Guide* contains guidance information about the EXEC CICS DOCUMENT commands.

Part 5. Using the CICS Transaction Gateway for OS/390

This part of the book describes the CICS Transaction Gateway for OS/390, and the CICS-provided Java classes for constructing ECI requests. It contains the following chapters:

Chapter 20. CICS Transaction Gateway overview

The IBM CICS Transaction Gateway provides secure, easy access from Web browsers and network computers to business-critical applications running on a CICS Transaction Server or TXSeries™ server using standard Internet protocols in a range of configurations.

The CICS Transaction Gateway is provided for the OS/2, Windows NT, AIX, and Solaris platforms. The CICS Transaction Gateway is also provided for Windows 95/98, but on these platforms it can only be used for development and not for production purposes.

CICS Transaction Gateway for OS/390 is provided for the OS/390 platform. However, unlike the other CICS Transaction Gateways, which can access multiple CICS servers, CICS Transaction Gateway for OS/390 can only access Transaction Server for OS/390.

The CICS Transaction Gateway for OS/2, Windows, AIX, and Solaris use a CICS Universal Client to route external call interface (ECI) and external presentation interface (EPI) requests to a CICS server (see “The external access interfaces (EPI and ECI)” on page 122). On the other hand, CICS Transaction Gateway for OS/390 can only route ECI requests and not EPI requests. The CICS Transaction Gateway for OS/390 actually uses the CICS External Call Interface (EXCI) to pass requests to CICS (see “CICS External Call Interface (EXCI)” on page 124). However, to a Java applet or application these appear to be ECI requests.

Figure 20 shows how a web-client can access CICS programs and data. Note that the CICS Transaction Gateway is shown as installed on a Web server machine. CICS Transaction Gateway need only be installed on the Web server machine if you are using the CICS Transaction Gateway with Java applets.

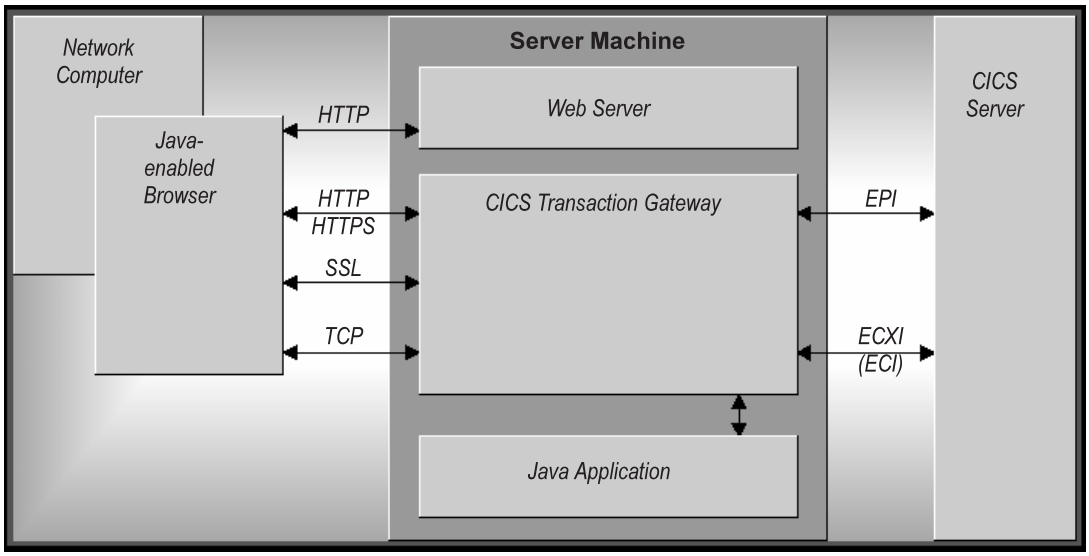


Figure 20. CICS Transaction Gateway

What the CICS Transaction Gateway for OS/390 provides

The CICS Transaction Gateway for OS/390 provides the following:

1. A **Java gateway application** that communicates with CICS applications running in CICS servers through the EXCI (External CICS Interface). This Java application was previously available in the IBM CICS Transaction Gateway for OS/390.
2. A **CICS Java class library** that includes classes that provide an application programming interface (API), and are used to communicate between the Java gateway application and a Java application (applet). The class **JavaGateway** is used to establish communication with the Gateway process, and uses Java's sockets protocol. The class **ECIRequest** is used to specify the ECI calls that are flowed to the gateway. These Java classes were previously available in the IBM CICS Transaction Gateway for OS/390.

The CICS Transaction Gateway for OS/390 can concurrently manage many communication links to connected Web browsers, The multithreaded architecture of the CICS Transaction Gateway for OS/390 enables a single Gateway to support multiple concurrently connected users.

Java

This section discusses the Java language, including the types of program that can be developed, and the security implications.

The Java language

The Java language can be used to construct Java *applets* and Java *applications.*

Java is an interpreted object-oriented language, similar to C++ that can be used to build programs that are platform-independent in both source and object form. Its unique operational characteristics, which span Web browsers as well as Web servers, enable new and powerful functions in Internet applications.

To achieve platform independence, the Java language allows no platform dependent-operations, and it excludes some C++ functions such as a preprocessor, operator overloading, multiple inheritances, and pointers. All Java programming is encapsulated within classes, and the Java Development Kit (JDK) includes special classes that are critical to assuring platform independence, and includes GUI functions, input/output functions, and network communications.

The Java compiler produces an intermediate bytecode format that is machine independent. This, in turn, is processed at execution time by a Java interpreter. The interpreter also inspects the bytecode at execution time to ensure its validity and safety to the machine environment. Because of the isolation the Java interpreter provides, it is sometimes referred to as a Java Virtual Machine.

Java applets

A Java applet is a small application program that is downloaded to, and executed on, a Java-enabled Web browser or network computer. A Java applet typically performs the type of operations that client code would perform in a client/server architecture. It edits input, controls the screen, and sends transactions to a server that performs data or database operations.

Applets are started using the <applet> HTML tag; this gives the applet control and specifies the display area to be used by the applet. When a Java-enabled server is

downloading a page and encounters this tag, it also downloads the applet bytecode, in the same way that it downloads an image that is referenced by an HTML image tag. The Java-enabled browser then interprets and executes the applet bytecode. The applet may edit screen input, generate screen output, and communicate back to the computer from which it was downloaded. Multiple applets can execute concurrently.

An example of applet processing is an applet in constant communication with a server to receive stock trade information, which it would update in a window on the screen.

The downloading of applets should not have a significant performance impact on the response time to end users since the applets are typically not very big. Applets may even improve overall browser performance by eliminating iterations with the Web server. Also, just as images are cached in Web browsers, applets are cached, minimizing the frequency of applet downloading.

Java applications

A Java application is a program that executes locally on a computer. It has platform-dependent capabilities in addition to those of an applet. It can access local files, create and accept general network connections, and call native C or C++ functions in machine-specific libraries.

Java applications can use the CICS-provided Java classes to perform transaction processing in CICS systems. They can use the **JavaGateway** class to establish two kinds of connection:

- A *network* gateway connection is a connection across a network to a CICS Transaction Gateway.
- A *local* gateway enables a Java application to communicate directly with a locally-installed CICS Transaction Gateway for OS/390, without the need for a network.

When the connection between the application and the CICS Transaction Gateway for OS/390 has been established, an application can use the **ECIRequest** class to do transaction processing. (For CICS Transaction Gateway for OS/390, the **EPIRequest** class cannot be used.)

Firewalls

A current design consideration in the use of Java applet communication is the impact of *firewalls*. This is the term given to a configuration of software that prevents unauthorized traffic between a trusted network and an untrusted network. Firewalls are put in place to protect company assets from outside intrusion, but they can also limit legitimate communications as well. Firewalls come into play in two ways:

1. The general accessibility of a server to outside users - inbound restrictions
2. The ability of end users inside a firewall to perform certain network functions outside their firewall - outbound restrictions.

A CICS Transaction Gateway for OS/390 configuration is well suited to avoid problems in the first area since the Gateway processor can be placed outside the firewall and be connected through the firewall to the CICS server. Outbound

firewalls that end users may have to contend with can be a problem. A large company might use a firewall to limit the types of connections and protocols that can be used.

The use of Java on an Intranet (a local implementation of the Internet) works very well since firewalls are typically not a factor. However, when designing Internet applications for end users outside a company, you should determine if end user firewalls will be an implementation factor. If so, then alternative processing for those users, such as executing the Java code as a Java servlet on the Web server, may be necessary. Also you should consider the use of the HTTP and HTTPS protocols supported by CICS Transaction Gateway for OS/390, see “Secure sockets layer (SSL)” on page 124 and “HTTPS” on page 125.

Web browsers and network computers

The CICS Transaction Gateway for OS/390 requires a Java-enabled Web browser, that is JDK version 1.1 enabled, for example, Netscape Navigator 2.0.2 or later, Microsoft Internet Explorer 4.0 or later. For more information, refer to “Web browsers” on page 127.

The Web browser communicates with the Web server using HyperText Transport Protocol (HTTP) requesting HyperText Markup Language (HTML) pages to be downloaded. These HTML pages can include calls to Java applets (see “Java applets” on page 118); multiple applets can run concurrently.

You may find that each browser displays the same information differently.

A network computer is a low-cost computer for the Internet user, it does the same things as a Web browser.

Web servers

A Web server is a software program that responds to information requests generated by Web browsers. When a request from a browser is received, the Web server processes the request to determine the action to take:

- Return the requested document
- Deny the request
- Pass the request through for further processing by an external application. The request might be, for example, to a database to perform a search request, or to a more dynamic form of information delivery such as Lotus Domino.

Communication between a Web server and an external application is transparent, you need to know only the URL of the Web server to direct a request to it. Also, all Web servers can handle requests from many browsers concurrently.

Specialized servers can also be configured to limit access to a restricted set of users, or to provide security for purchase of goods or services.

Web servers exist for almost every platform and are available from many suppliers. For information on the Web servers supported by CICS Transaction Gateway for OS/390, see “Web servers” on page 127.

How the CICS Transaction Gateway for OS/390 accesses CICS

This section describes how the CICS Transaction Gateway for OS/390 allows access to CICS programs and data.

Figure 21 shows the flow of control when a Web browser calls CICS transaction processing facilities using the CICS Transaction Gateway for OS/390.

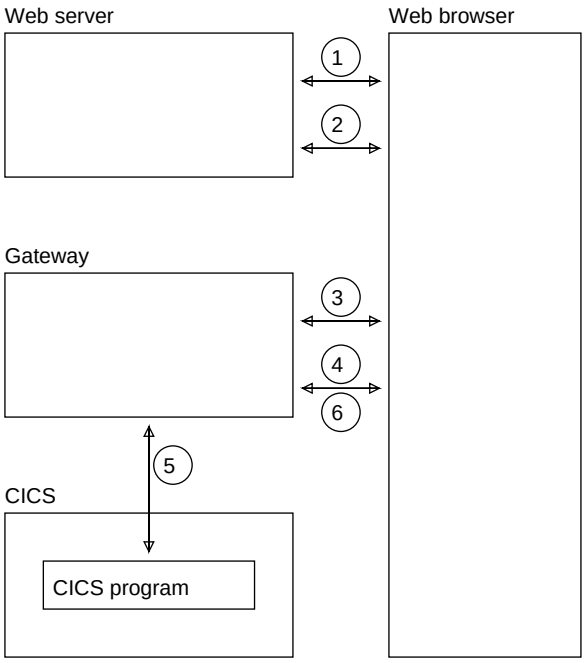


Figure 21. Flow of control with the CICS Transaction Gateway for OS/390

1. The Web browser calls the Web server using HTTP (Hypertext Transfer Protocol) to get HTML pages.
2. When the browser, which is interpreting the HTML and presenting it on the screen, finds an applet tag, it calls the Web server to get the applet and the classes that it needs. It then executes the applet.
3. An applet that is going to communicate with CICS creates a **JavaGateway** object. The creation of this object causes a call to the CICS Transaction Gateway for OS/390 long-running task.
4. The applet creates an **ECIRequest** object to represent its request for a CICS program, and calls the **flow** method of the **JavaGateway** object, passing the instance of the **ECIRequest** object.
5. The CICS Transaction Gateway for OS/390 receives the request, and calls the CICS program.
6. When the CICS program ends, the results are returned to the Web browser.

Figure 22 on page 122 shows the flow of data when a Web browser calls CICS transaction processing facilities using the gateway.

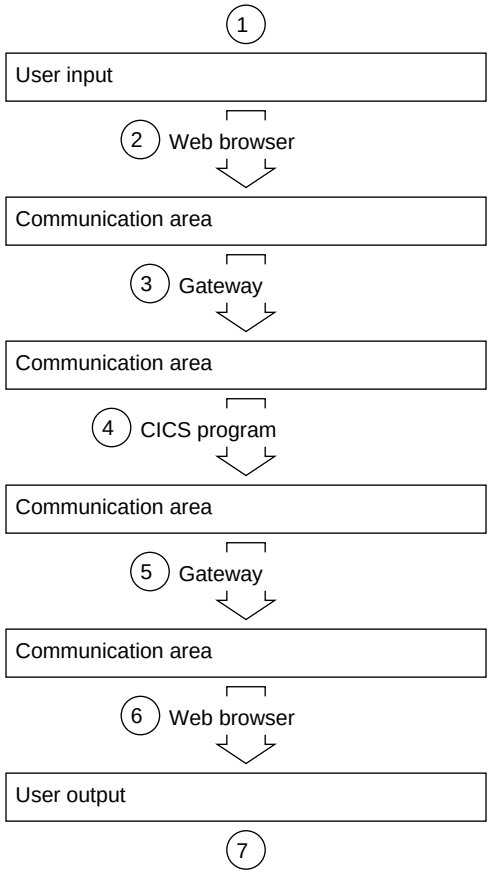


Figure 22. Data flow with the CICS Transaction Gateway for OS/390

1. The Web browser acquires data for the CICS program from the end user.
2. The Web browser constructs a communication area for the CICS program that is to supply transaction processing services.
3. The CICS Transaction Gateway for OS/390 receives the communication area and passes it to the CICS program. The contents of the communication area are translated from ASCII (in the gateway) to EBCDIC (in the CICS Transaction Server).
4. The CICS program supplies the transaction processing services, enquiring on and perhaps changing CICS resources. If the program ends normally, changes to recoverable resources are committed. If the program ends abnormally, the changes are backed out.
5. The communication area is translated from EBCDIC to ASCII, and returned by the gateway to the Web browser.
6. The Web browser presents information to the end user.

The external access interfaces (EPI and ECI)

The external access interfaces allow non-CICS applications to access and update CICS resources by initiating CICS transactions, or by calling CICS programs. When used in conjunction with the CICS communication facilities, they enable non-CICS programs to access and update resources on any CICS system. This supports such

activities as: developing graphical user interface (GUI) front ends for CICS applications, and allowing the integration of CICS systems and non-CICS systems.

The external presentation interface (EPI) allows you to develop GUIs, either for existing CICS systems or for new applications. It is particularly useful for developing new GUI front ends for existing CICS transactions, which need not be changed. The application can use the EPI to communicate with a CICS transaction, and can exploit the presentation facilities of the client system to communicate with the end user.

The external presentation interface (EPI) is **not** supported by the CICS Transaction Gateway for OS/390.

The integration of CICS and non-CICS systems usually involves passing user-defined data between the programs of the non-CICS system and a CICS program, and the external call interface (ECI) can be used for this.

External Call Interface (ECI)

The ECI allows a non-CICS application to call a CICS program in a CICS server. The application can be connected to several servers at the same time, and it can have several program calls outstanding at the same time.

The CICS program cannot perform terminal I/O, but can access and update all other CICS resources. The same CICS program can be called by a non-CICS application using the ECI, or by a CICS program using EXEC CICS LINK. Data is exchanged between the two programs by means of a COMMAREA, in a similar way to CICS. The user can specify the length of the COMMAREA data to optimize performance.

Calls may be made synchronously or asynchronously. Synchronous calls return control when the called program completes, and the information returned is immediately available. Asynchronous calls return control without reference to the completion of the called program, and the application can ask to be notified when the information becomes available.

Calls may also be *extended*. That is, a single logical unit of work may cover two or more successive calls, though only one call can be active for each logical unit of work at any time. If it uses asynchronous calls, the application can manage many logical units of work concurrently.

The called program can update resources on its own system, it can use distributed program link (DPL) to call CICS programs on other systems, and it can access resources on other CICS systems by function shipping, by distributed transaction processing (DTP), or (in the Transaction Server for OS/390 environment) by the front end programming interface (FEPI).

For more information on the external access interfaces, see *CICS Family: Client/Server Programming*.

CICS External Call Interface (EXCI)

The external CICS interface (EXCI) is analogous to the ECI, and is an application programming interface that enables a program running in MVS, such as the CICS Transaction Gateway for OS/390, to call a program running in a CICS region and to pass and receive data by means of a communication area.

The EXCI allows a user to allocate and open sessions (or pipes) to a CICS region, and to pass distributed program link (DPL) requests over them. The multiregion operation (MRO) facility of CICS interregion communication (IRC) facility supports these requests, and each pipe maps onto one MRO session, with a limit of 100 pipes per EXCI address space.

For more information on the EXCI, refer to the *CICS External Interfaces Guide*.

Security

The CICS Transaction Gateway for OS/390 provides comprehensive support for secure communication, which is critical to successful Internet operation:

Privacy
Provided via implementations of the security protocols secure sockets layer (SSL) and HTTPS (HTTP over SSL), which encrypt data flowing between the browser and the Gateway server.

Authentication
Provided via SSL (″server-only″ authentication).

Authorization
Provided as a standard function of each CICS server, enabling customers to control what transactions a given user can run and what data he or she can access.

Also, Security Exits are provided that enable the user to define security operations such as public key encryption. They may also be used for data compression. Some example source files that demonstrate these functions are provided.

Secure sockets layer (SSL)

SSL is a **Handshake Protocol** developed to provide security and privacy over the Internet. The SSL protocol uses encryption and authentication to ensure:

privacy
The data to be exchanged between the client and the server is encrypted, so that only that client (your application) and that server (the CICS Transaction Gateway for OS/390) can make sense of the data.

SSL uses public key encryption as a secure mechanism to distribute a secret key between the server and the client. Public key encryption is a technique that uses a pair of asymmetric keys for encryption and decryption. In the case of SSL, a secret (symmetric) key is passed between the client and server (using public key cryptography), which is then used to encrypt and decrypt all traffic along the SSL connection. This encryption protects the data from other parties trying to eavesdrop, as no other parties will have the secret key needed to decrypt the data. This ensures that private information such as a credit card number is transferred securely.

integrity
The message transport includes a message integrity check based on a secure hashing algorithm. This algorithm is performed when the message is sent, and again when it is received. If the two hash values do not match, the receiver is warned that the message may have been tampered with.

authentication
The CICS Transaction Gateway's implementation of the SSL protocol provides ″server-only″ authentication. When a client establishes a connection with the CICS Transaction Gateway, it is required to authenticate the server's details. The authentication mechanism is based on the exchange of digital certificates (X.509v3 certificates). These digital certificates contain information about an entity, like the system name and public key, and the server's digital signature. Digital certificates are issued by a Certificate Authority (CA), and encrypted using the CA's private key. If you can decrypt the certificate using the CA's public key, you know that the information contained within the certificate can be trusted, that is, that the certificate really does belong to whoever claims to own it.

HTTPS

HTTPS is a variant of HTTP for handling secure transactions. Most current browsers support a URL access method, "HTTPS" for connecting to HTTP servers using SSL. HTTPS is a unique protocol that is simply SSL underneath HTTP. A secure connection is typically made with a URL similar to "https://someAddress" The default HTTPS port number is 443, as assigned by the Internet Assigned Numbers Authority.

Keys and Certificates

The SSL protocol uses Public-Key cryptography, which has been recommended for use with the ISO authentication framework, also known as the X.509 protocol. This framework provides for authentication across networks.

The most important part of X.509 is its structure for public-key certificates. A trusted Certification Authority (CA) assigns a unique name to each user and issues a signed certificate containing the name and the user's public key.

The CICS Transaction Gateway for OS/390 allows you to establish yourself as a Certificate Authority to allow you to issue ″self-signed″ X.509 certificates, removing the need for an external CA signer. These X.509 certificates are then "encapsulated" into a Java classfile, which can then be used by the SSL and HTTPS protocols. These Java classfiles are referred to as *KeyRing* classes.

For information on generating your own KeyRing classfiles, for both the client and server, refer to "Creating and configuring KeyRings" on page 139.

Chapter 21. CICS Transaction Gateway for OS/390 planning

This chapter helps you plan the installation of the CICS Transaction Gateway for OS/390 by discussing the software requirements, the Web Servers and browsers that are supported, and also migration issues.

Software requirements

For the development of Java applets and applications, **Java Development Kit (JDK)** Version 1.1.6 is required.

Note: JIT compilers may cause problems with the HTTPS and SSL protocols and therefore should be disabled if you wish to use these secure protocols.

Web servers

CICS Transaction Gateway for OS/390 supports the following Web servers:

- Lotus Domino Go Webserver

Web browsers

The CICS Transaction Gateway for OS/390 should work with any JDK Version 1.1 compliant Java-enabled browser. This includes the following:

- OS/2: Netscape 2.0.2 with 1.1 patch
- Windows NT, 98, 95: Microsoft Internet Explorer 4.0.1
- Windows NT, 98, 95: Netscape 4.0.3 with 1.1 patch
- AIX: Netscape 4.0.4
- Solaris: HotJava Browser Version 1.1

You can also use the JDK AppletViewer to run applets.

Restrictions

There is a compatibility problem between Internet Explorer Version 4 (or later) and the CICS Transaction Gateway's HTTPS protocol. This means that it is not possible to establish a https: connection to a CICS Transaction Gateway from an applet running within Internet Explorer's Java Virtual Machine.

Migrating from CICS Transaction Gateway for OS/390

If you are a user of CICS Transaction Gateway for OS/390 migrating to CICS Transaction Gateway for OS/390, you may have customized the Gateway.properties file or the JGate script to set environment variables. If you have customized these files, you should save a copy of them before you install CICS Transaction Gateway for OS/390.

Due to the renaming of the CICS Transaction Gateway for OS/390 package, you must change all `import` statements in your programs and recompile them. Therefore you must change all occurrences of:

`import ibm.cics.jgate.client.*` to: `import com.ibm.ctg.client.*`

| `import ibm.cics.jgate.security.*` to: `import com.ibm.ctg.security.*`

| Due to a change in the ClientSecurity and ServerSecurity interfaces, any user
| classes that implement these methods need to be changed. The methods called to
| generate handshake data are now passed the TCP/IP address of who they are
| handshaking with. Also, an AfterDecode method has been added to both interfaces.

Chapter 22. Installing CICS Transaction Gateway for OS/390

This chapter describes how to install CICS Transaction Gateway for OS/390.

Installation

The CICS Transaction Gateway for OS/390 is provided as the file CTG-302betamvs.tar.Z, which contains the CICS Transaction Gateway for OS/390 software that is to be installed in the hierarchical file system (HFS).

This material must first be copied into a directory on a workstation that can communicate with the System/390 system on which you wish to install the CICS Transaction Gateway for OS/390.

When you have copied the program material to the workstation, perform the following actions:

1. Transfer the program material from the workstation into a suitable directory in HFS as follows:
   a. Upload the file to an MVS sequential data set. You can use your terminal emulator transfer options to do this.
   b. Use the OPUT command on TSO to put the data set in HFS:

      ```
      OPUT 'CTG-302BETAMVS.TAR.Z' '/pathCTG-302betamvs.tar.Z' BINARY
      ```

      where *path* is the HFS path that is to be the root for the CICS Transaction Gateway for OS/390 directory structure.
2. Use the OMVS command on TSO to enter the OS/390 UNIX System Services shell. Then use the command

   ```
   cd /path
   ```

   to change to the directory into which you put CTG-302betamvs.tar.Z.
3. Use the command

   ```
   uncompress CTG-302betamvs.tar.Z
   ```

   to uncompress the file.
4. Use the command

   ```
   tar -xopfv CTG-302betamvs.tar
   ```

   where:
   - `-x` specifies that all files are extracted
   - `-o` allows the original file permission bits for owner, group and all to be restored. This is useful for all the execute bits.
   - `-p` does not try to restore the original owner and group id when extracting files. The default is to do this, but most likely the person doing the extract will not have permission, or the user id will not exist on the system.
   - `-f` Specifies the file name of the archive file.
   - `-v` specifies verbose messages (this is optional).

   This creates the directories illustrated in Table 1 on page 130.

Note: To transfer the program material you can also use other methods, for example, using ftp.

Installing CICS Transaction Gateway for OS/390

Table 1. Contents of directories

Relative path	Contents.
(root)	Download directory.
/ctg	Installation directory. Contains the readme.txt file.
/ctg/bin	CICS Transaction Gateway for OS/390 executable code.
/ctg/classes/com/ibm/ctg/client	Java classes for use by client applications.
/ctg/classes/com/ibm/ctg/server	Java classes for use by the CICS Transaction Gateway for OS/390.
/ctg/classes/com/ibm/ctg/security	Java security classes for use by the CICS Transaction Gateway for OS/390.
/ctg/classes/com/ibm/ctg/test	Java classes for the TestECI program.
/ctg/html/doc	Documentation for the CICS Transaction Gateway.
/ctg/samples/java/com/ibm/ctg/test	Java source files for samples.

CICS definitions for the CICS Transaction Gateway for OS/390

For Version 1.2 only, you must start CICS Transaction Server with the system initialization parameter WEB=YES to use the CICS Transaction Gateway for OS/390. The default value for this parameter is WEB=NO. For more information, refer to the *CICS System Definition Guide*.

You must define the program DFHJVCVT to CICS Transaction Server. The group DFHJAVA contains the program definition, and you can put it in the startup list, or install it with the CICS CEDA command. For more information, refer to the *CICS System Definition Guide*.

The CICS Transaction Gateway for OS/390 needs a session definition and a connection definition for EXCI. For details of these definitions see the *CICS External Interfaces Guide*.

Note: If you use the sample DFH$EXCI, you must set the DFHJVPIPE environment variable to BATCHCLI in order to use the specific pipe. If you do not set the variable, the generic pipe is used. For more information see “Environment variables” on page 131.

Chapter 23. Configuring CICS Transaction Gateway for OS/390

This chapter describes what you can do to configure CICS Transaction Gateway for OS/390:

- Set properties of the CICS Transaction Gateway for OS/390 in the Gateway.properties file.
- Use environment variables, for example, to control the use of EXCI pipes, and the names of CICS servers that callers can access.
- Edit the JGate script that is used to start CICS Transaction Gateway for OS/390.

Environment variables

You can set the following environment variables for CICS Transaction Gateway for OS/390:

CLASSPATH
The path in HFS for the CICS-provided Java class definitions, that is, /ctg/classes. You **must** set CLASSPATH to compile and run Java applications. You should concatenate the existing CLASSPATH to the new CLASSPATH:

```
export CLASSPATH=/ctg/classes:$CLASSPATH
```

DFHJVPIPE
The name of the specific pipe that the CICS Transaction Gateway for OS/390 is to use for EXCI calls. If this variable is not set, the CICS Transaction Gateway for OS/390 uses a generic pipe.

DFHJVSYSTEM_*nn*
You may set up to 100 environment variables of the form DFHJVSYSTEM_*nn*, where *nn* ranges from 00 to 99. The first variable must be DFHJVSYSTEM_00, and subsequent variables must have consecutive numbers, or they will not be recognized. The value of the variable is the name and description of a CICS system to be returned in response to a request for the **CICS_EciListSystems** function. The value must be a string containing the uppercase APPLID of the CICS system immediately after the equals sign (=), followed by a hyphen (-), followed by a description of the CICS system, which must not be more than 60 bytes long. For example if you specify:

```
DFHJVSYSTEM_00=MYCICS-Test CICS system
```

CICS_EciListSystems returns details of a CICS system called MYCICS with description `Test CICS system`.

LIBPATH
The path in HFS for the CICS Transaction Gateway for OS/390 executable code. You must set this variable to `/ctg/bin`. You should concatenate the existing LIBPATH to the new LIBPATH:

```
export LIBPATH=/ctg/bin:$LIBPATH
```

STEPLIB
The library containing the default EXCI options and the EXCI load modules. This must be the same as specified for EXCI_LOADLIB in the JGate script, see “Customizing the JGate script” on page 136.

You can set the environment variables in three ways:

Configuring CICS Transaction Gateway for OS/390

1. By entering commands on an OS/390 UNIX System Services command line. For example, the following command, issued in the /ctg/bin directory, sets the DFHJVPIPE environment variable to use a specific pipe called JAVAGAT1.

   ```
   export DFHJVPIPE=JAVAGAT1
   ```

   If the value of an environment variable contains spaces, it must be enclosed in single or double quotes. For example:

   ```
   export DFHJVSYSTEM_00="MYCICS-Test CICS system"
   ```

2. By editing the JGate script that is used to start the gateway. See “Customizing the JGate script” on page 136.
3. By creating JCL to start the gateway that includes environment variable settings. See “Starting with JCL” on page 146.

Configuring the Gateway.properties file

The Gateway.properties file allows you to set persistent properties for the CICS Transaction Gateway. These properties are read when the Gateway is started, and can be broadly split into the following categories:

- **General start up properties**.

  These properties are those that can also be specified via command line options when the Gateway is started, see “Starting the Gateway with user-specified options” on page 145.

- **Network protocol handler properties**.

  These properties define which network protocol handlers are started. The Gateway supports dynamic protocol handler loading, and so additional protocol handlers can be added by adding entries in the properties file. Also, protocol handler specific parameters can be specified.

- **Additional properties**

  These properties can only be set in the Gateway.Properties file, and not via command line options.

The Gateway.properties file is located in the /ctg/bin directory.

The Gateway.properties file takes the form of a standard Java properties file, with a particular property defined by a line of the form:

```
property=value
```

Any line that starts with a # character is treated as a comment.

General start up properties

It is possible to provide preset values for any property that can be specified via a Gateway command line option. Currently, the following general properties are allowed in the Gateway.properties file.

port=*port_number*
: The TCP/IP port number for the tcp: protocol.

initconnect=*number*
: The initial number of ConnectionManager threads.

maxconnect=*number*
The maximum number of ConnectionManager threads. If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

initworker=*number*
The initial number of Worker threads.

maxworker=*number*
The maximum number of Worker threads. If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

trace=[on|off]
Enables or disables extra tracing messages.

time=[on|off]
Enables or disables timing information in messages.

noinput=[on|off]
Enables or disables the reading of input from the console.

nonames=[on|off]
Enables or disables the display of TCP/IP hostnames.

If a property defined in the Gateway.properties file is specified via the associated command line option when the Gateway is started, the command line setting ***takes precedence***.

Network protocol handler properties

Protocol handler entries in the Gateway.properties file determine which network protocol handlers are loaded by the Gateway when it starts. This allows for the Gateway to be expanded to support new network protocols in the future, by simply adding a new Java class along with the relevant entry in the Gateway.properties file.

Entries for a particular protocol begin with protocol@*protocol-name*. followed by which property is being set. You can set the following properties:

1. protocol@*protocol-name*.handler = ...
   The name of the class that provides the handler for this protocol
2. protocol@*protocol-name*.parameters = ...
   The parameters passed to the protocol handler. The possible parameters vary upon the protocol handler, but generally take the form of a semi-colon separated list. If a particular parameter is not specified then a default value is used.

So, for example, the entries for the standard tcp: protocol handler are :

```
protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
protocol@tcp.parameters=port=2345;sotimeout=1000;connecttimeout=1000;idletimeout=600000;pingfrequ
```

The following protocol handlers are supported :

- **tcp:** - com.ibm.ctg.server.TCPHandler, which supports the following parameters:

  port=*TCP/IP port*
  The TCP/IP port number on which this protocol accepts connections. This value is overridden by the -port command line option.

  sotimeout=*period in milliseconds*
  How frequently the tcp: handler wakes from accepting inbound connections. When it wakes, it checks to see whether the Gateway is being stopped, and

so this value affects the time taken for the Gateway to shutdown cleanly. If you set this value to zero then the handler only wakes when a new connection is accepted, and so the Gateway will not shutdown cleanly until that time.

connecttimeout=*period in milliseconds*
When a new connection has been accepted, this value specifies how long the protocol handler waits for a ConnectionManager thread to become available. If a ConnectionManager thread does not become available, then the connection is refused. If this value is set to zero, a connection is refused if a ConnectionManager is not immediately available.

idletimeout=*period in milliseconds*
Specifies how long a connection is allowed to remain dormant. The idle timeout period is counted from when a request was last flowed down the connection. When the idle timeout has expired, the connection is disconnected, though if work is still in progress on behalf of the connection, it may be left connected. If this value is not set, or is set to zero, then idle connections will not be disconnected.

pingfrequency=*period in milliseconds*
Specifies how often a PING message is sent by the Gateway to an attached client to check that client is still active. If a PONG response has not been received by the time the next PING message is due to be sent, then the connection is disconnected. Again, if work is still in progress on behalf of the connection it may be left connected. If this value is not set, or is set to zero, then PING messages are not sent.

dropworking
Specifies that a connection can be disconnected, due to an idle timeout or a PING/PONG failure ***even*** if work is still in progress on behalf of this connection.

solinger=*period in milliseconds*
Specifies the SO_LINGER setting for any Socket used by this handler. If this value is not set or set to zero, then SO_LINGER is disabled for any Sockets used by this handler.

requiresecurity
Enables the requiresecurity property for this protocol. See page 135 for a description of the requiresecurity option and how to enable it globally for all protocols.

- **SSL:** - com.ibm.ctg.server.SslHandler, which supports exactly the same parameters as tcp (above), with the exception of:

port=*TCP/IP port*
The TCP/IP port number on which this protocol accepts connections. The default port is 8050.

keyring=*CLASSname*
Specifies the CLASSname of the server KeyRing. The server KeyRing should consist of a valid x.509 certificate that is used to identify this server to connecting clients. This KeyRing class is generated using the SSL tools supplied with this product. For more information about SSL and its associated KeyRing classes, see “Chapter 24. CICS Transaction Gateway for OS/390 security” on page 139.

keyringpw=*PASSword*
The Password of the specified server KeyRing class.

- **HTTP:** - com.ibm.ctg.server.HttpHandler, which supports the following parameters:

  port=*TCP/IP port*
  The TCP/IP port number on which this protocol accepts connections. Unlike the tcp: protocol, this value can only (and **must**) be specified here. The default port is 8080.

  sotimeout=*period in milliseconds*
  As for the tcp: protocol.

  connecttimeout=*period in milliseconds*
  As for the tcp: protocol.

  idletimeout=*period in milliseconds*
  As for the tcp: protocol.

  dropworking
  As for the tcp: protocol.

  solinger=*period in milliseconds*
  As for the tcp: protocol.

  requiresecurity
  As for the tcp: protocol.

- **HTTPS:** - com.ibm.ctg.server.HttpsHandler, which supports the same parameters as HTTP (above), with the exception of

  port=*TCP/IP port*
  The TCP/IP port number on which this protocol accepts connections. The default port is 443.

  keyring=*CLASSname*
  Specifies the CLASSname of the server KeyRing. The server KeyRing should consist of a valid x.509 certificate which is used to identify this server to connecting clients. This KeyRing class is generated using the SSL tools supplied with this product. For more information about SSL and its associated KeyRing classes, see “Chapter 24. CICS Transaction Gateway for OS/390 security” on page 139.

  keyringpw=*PASSword*
  The Password of the specified server KeyRing class.

Additional properties

You can set the following additional properties:

workertimeout=*period in milliseconds*
When a ConnectionManager accepts a request, it must allocate a Worker thread to execute that request. If however, a Worker does not become available within the workertimeout period an error message is sent rejecting that request and the request is not executed. By default, this timeout is set to 1000 milliseconds, but you may set this property to override that default. If this value is set to zero, the request is rejected, if a Worker is not immediately available.

closetimeout=*period in milliseconds*
When a Java-client program disconnects from the Gateway, the Gateway may still be processing requests on behalf of that program. If work is still in progress, the ConnectionManager that was managing requests on behalf of that Java-client, waits for in-progress requests to complete for up to the closetimeout period. If after this period there are still requests in progress, the

ConnectionManager continues its processing and marks itself as available for use by a new connection. By default this timeout is set to 5000 milliseconds, but you may set this property to override that default. If this value is set to zero, the ConnectionManager does not wait for in progress requests to complete.

requiresecurity=yes
When a Java-client program connects to the Gateway, it can specify a pair of security classes that should be used on the connection. However, by default a Gateway also accepts connections from programs that do not specify this pair of security classes. If you only wish your Gateway to accept connections that use security classes you should set requiresecurity=yes. This setting affects all protocols. You can control which security classes are valid by controlling the set of ServerSecurity classes that can be accessed by your Gateway.

Customizing the JGate script

The JGate script that is used to start the CICS Transaction Gateway for OS/390 is supplied in the `/ctg/bin` directory.

You may choose to customize the EXCI options used by the CICS Transaction Gateway for OS/390 by tailoring the EXCI options table, DFHXCOPT. In this case, you must update the EXCI_OPTIONS specification in the JGate script to specify the library that contains the customized options table. Find the line:

```
EXCI_OPTIONS="your.user.loadlib"
```

and change it to specify the appropriate library. For information about the EXCI options and how to customize them, see the *CICS External Interfaces Guide*.

Note: If the CICS region has no security (SEC=NO), the DFHXCOPT table option SURROGCHK=NO must be specified.

You must update the high-level qualifier for the name of the library specified in EXCI_LOADLIB in the JGate script. This library contains:

- The default EXCI options.
- The EXCI load modules.

Find the line:

```
HLQ="CICSTS13.CICS"
```

and change it to specify the high-level qualifier of your CICS installation.

You may add export commands to the JGate script to set the values of environment variables for the CICS Transaction Gateway for OS/390, which are described in “Environment variables” on page 131. If you set environment variables in the JGate script, these override settings in the JCL.

Other configuration tasks:

To complete the configuration of CICS Transaction Gateway for OS/390, you must perform the following:

- Configure CICS for the EXCI requests from the CICS Transaction Gateway for OS/390. Information about this task is given in the *CICS External Interfaces Guide*.

- Create conversion templates to be used to translate the communication area used by the CICS programs requested by Web browsers. Information for this task is given in *CICS Family: Communicating from CICS on System/390*.

Configuring CICS Transaction Gateway for OS/390

Chapter 24. CICS Transaction Gateway for OS/390 security

This chapter describes the configuration you need to do before using the security protocols SSL and HTTPS.

Creating and configuring KeyRings

The CICS Transaction Gateway for OS/390 uses *KeyRings* to hold X.509 certificate information for use when establishing SSL and HTTPS connections.

A trusted Certification Authority (CA) assigns a unique name to each user and issues a signed certificate containing the name and user's public key. The CICS Transaction Gateway for OS/390 provides a mechanism for you to "self-sign" your certificates. You establish yourself as your own Certification Authority and generate the X.509s for the server (CICS Transaction Gateway for OS/390) and client (browser) side.

To create your own KeyRing, you must:

1. Generate a public and private keypair.
2. Create a self-signed CA certificate.
3. Create a Vault object to hold the server private key and server certificate together.
4. Embed the certificates into Java class files.

Note: The CICS Transaction Gateway does not provide support for importing externally-signed digital certificates.

Generating a public and private keypair

Before you issue the commands described in the following sections, you should set CLASSPATH as described in "Environment variables" on page 131.

You must first generate a key pair for use by the server. The keypair consists of a public and private key. Enter the following command:

```
java com.ibm.cfwk.tools.KeyGenTool --forge "RSA/512/F4" server.key
```

This generates an RSA type key pair, of length 512 bits, and with the public exponent being the number Fermat 4 (X'10001'). The resultant key pair is stored as a file (server.key) in the working directory.

Creating a self-signed CA certificate

Next, create a self-signed CA certificate. Enter the following command, with *all the parameters on one line*:

```
java com.ibm.cfwk.tools.MakeCertTool
     --serial 0 --for 1y
     --issuer "cn=Server Name, ou=Organisational Unit, o=Organisation, c=Country"
     --sign-alg "MD5 with RSA"
     --sign-key server.key
     --cert-file server.cer
```

where:

serial
is the serial number of the certificate, specified as zero (root)

for
is the validity period (1 year).

issuer
This parameter consists of a number of sub-fields. When the default ClientKeyRing and ServerKeyRing classes supplied with CICS Transaction Gateway for OS/390 were generated, the subfields were as follows:

```
--issuer "cn=CICS Transaction Gateway, ou=IBM Hursley Labs., o=IBM UK, c=GB"
```

sign-alg
is the signing algorithm. This must always be specified as `MD5 with RSA`.

sign-key
is the previously generated server.key file.

cert-file
is the output file for the certificate, specified as server.cer in the working directory.

Creating a Vault object

When you have created a server CA certificate, you next need to create a **Vault** object to hold the server private key together with the generated server certificate. Enter the following command:

```
java com.ibm.cfwk.tools.VaultTool --password "serversecret" server.vault
               add private key K1 00-hex server.key
```

where:

- the password of the vault is specified as `serversecret`
- the server vault is to be stored in the working directory as `server.vault`.
- the private key `server.key` is to be added to the server vault file.
- K1 is an arbitrary label used to associate keys, certificates and certificate chains.

The vault can also store additional information in the form of a byte array. This feature is not used in the CICS Transaction Gateway for OS/390, and so a dummy byte array, (X'00') is specified.

When the server.key has been successfully added to the server.vault, add and associate the server certificate to the server.vault as follows:

```
java com.ibm.cfwk.tools.VaultTool --password "serversecret" server.vault
               add public chain K1 00-hex server.cer
```

With the server-side vault completed, you must now generate the vault object for the client. You use a similar process:

```
java com.ibm.cfwk.tools.VaultTool --password "clientsecret" client.vault
               add public cert XX 00-hex server.cer
```

where:

- the client vault's password is specified as `clientsecret`
- the resultant Vault object is stored as client.vault
- the server certificate is added to the client vault as a public certificate, meaning the client can "trust" the server to which the server.cer relates.

- an arbitrary label is used (XX) and a dummy byte array (X'00') is passed to the Vault object.

By this stage you have created both client and server X.509 certificates, which can be recognized by the SSL/HTTPS protocols.

Embedding the certificates into Java class files

The next step is to "embed" these certificates into Java classfiles, which can then be used by the CICS Transaction Gateway for OS/390. Use the following command:

```
java com.ibm.cfwk.tools.VaultTool --password "serversecret" server.vault
     container -p "serverkeyringsecret" ServerKeyRing
```

where:

- server.vault is accessed using the password `serversecret`
- server.vault's internal data is stored in the container `ServerKeyRing`
- ServerKeyRing is encrypted using the password `serverkeyringsecret`.

The container is simply a Java classfile, which the CICS Transaction Gateway refers to as a *KeyRing*.

Repeat the process for the client.vault, as follows:

```
java com.ibm.cfwk.tools.VaultTool --password "clientsecret" client.vault
                container -p "clientkeyringsecret" ClientKeyRing
```

where:

- client.vault is accessed using the password `clientsecret`
- client.vault's internal data is stored in the container, that is, the Java classfile `ClientKeyRing`
- ClientKeyRing is encrypted using the password `clientkeyringsecret`.

Cleaning up

When you have generated the client and server KeyRings, it is recommended that you delete the server.key and server.vault files from your hard disk. These files have the server's private key embedded in them, and are in any case no longer required.

Restricting access to the server Keyring

The contents of the server KeyRing are password encrypted; however it is highly recommended that you:

- ensure correct file permissions are in place
- restrict access, where applicable, to the CICS Transaction Gateway machine

Using SSL and HTTPS

The CICS Transaction Gateway for OS/390 provides two default KeyRing class files that can be used to establish SSL and HTTPS connections. The **ClientKeyRing** and **ServerKeyRing** are both encrypted using the password **default**, and are only recommended for use in testing environments.

Therefore, before you use the SSL and HTTPS protocols, we recommend that you follow the steps detailed in “Creating and configuring KeyRings” on page 139 to generate your own X.509 certificates.

The CICS Transaction Gateway for OS/390 supports various protocols for communicating with its clients, for example, TCP and HTTP, and these are defined in the Gateway.properties file, (see “Configuring the Gateway.properties file” on page 132). To use the SSL and HTTPS protocols, add the following handler entries:

```
protocol@ssl.handler=com.ibm.ctg.server.SslHandler
```

and

```
protocol@https.handler=com.ibm.ctg.server.HttpsHandler
```

These entries will start the SSL and HTTPS protocols when the CICS Transaction Gateway for OS/390 is executed. You can specify that just one of the security protocols is used, or both of them. With these handlers specified, when the Gateway is started, it listens for SSL requests on port 8050, and for HTTPS requests on port 443.

Like other protocols, SSL and HTTPS have various parameters that are set in the second part of the Gateway.properties file entry. Two important parameters that are specific to the SSL and HTTPS protocols are:

keyring=
: This parameter specifies the name of the server-side KeyRing classfile. CLASSPATH must be set so that this class can be found.

keyringpw=
: This parameter specifies the password used to ”decipher” the enciphered server KeyRing.

Full examples of SSL and HTTPS protocol entries are given in the Gateway.properties file (see “Configuring the Gateway.properties file” on page 132).

Which secure protocol is used will determine whether a client-side KeyRing is required. The HTTPS protocol is designed for secure communication from within a Java applet, where the browser (client) itself has the necessary functionality to establish a secure connection with the CICS Transaction Gateway for OS/390 (server). For this reason the HTTPS protocol only requires a server-side KeyRing to be specified, the client side is handled by the browser software.

The SSL protocol is designed at a much lower level in which the CICS Transaction Gateway for OS/390 has code to handle the server and the client in a secure fashion. The SSL protocol requires a KeyRing classfile for both the server *and* the client.

The client-side KeyRing is specified by setting a static field in the SslJavaGateway.class. This class forms part of the CICS Transaction Gateway for OS/390 client-side code.

The SslJavaGateway.class provides two methods: one for ”getting” and one for “setting” the client KeyRing:

```
public static void setKeyRing(String strSetKeyRing, String strSetKeyRingPW)
public static String getKeyRing()
```

To set the client KeyRing class to be used by the SSL protocol, your client application or applet would make a static call to the following method:

SslJavaGateway.setKeyRing(CLASSname, PASSword);

where:

- CLASSname denotes the classname of the Java KeyRing class generated for the client
- PASSword is used to decipher the embedded X.509 certificate.

The SslJavaGateway.class also provides the “getter” method **getKeyRing()** to return the CLASSname of the currently specified client KeyRing.

Using the SSL/HTTPS protocols to establish a connection from a client application or applet to the CICS Transaction Gateway for OS/390 is no different from using the TCP or HTTP protocols. The client application or applet simply “flows” its request to the CICS Transaction Gateway for OS/390 using the relevant URL. For example, for SSL the application would use `ssl://transGatewayMachine:8050`, or for HTTPS it would use `https://transGatewayMachine:443`.

See “Chapter 26. CICS Transaction Gateway for OS/390 programming overview” on page 149 and the CICS Transaction Gateway for OS/390 programming interface HTML pages for further information regarding the design and implementation of client-side programs.

Using the CICS Transaction Gateway for OS/390

Chapter 25. CICS Transaction Gateway for OS/390 operation

This chapter describes how to start and stop the CICS Transaction Gateway for OS/390. The startup options that you can specify are described.

Starting the CICS Transaction Gateway for OS/390

You can start the gateway:

- From an OS/390 UNIX System Services command line
- By submitting JCL to start a batch job

You can start the gateway with default values for the options, or with user-supplied values. The options and their values are described in "Starting the Gateway with user-specified options".

Starting from a command line

To start the CICS Transaction Gateway for OS/390 with the default options, type `JGate` at the command prompt and press Enter. You see the startup message:

```
CCL6500I: Starting the Gateway with default values.
```

This is followed by two lines showing the values that are being used, for example:

```
CCL6502I: [ Port = 2006 , Initial Connections = 1 , Maximum Connections = 100 ,
CCL6502I:   Initial Workers = 1 , Maximum Workers = 100 ]
```

Starting the Gateway with user-specified options

The user definable options on the start command are:

-port=*port_number*
The TCP/IP port number assigned to the CICS Transaction Gateway for OS/390

-initconnect=*number*
The initial number of ConnectionManager threads.

-maxconnect=*number*
The maximum number of ConnectionManager threads. If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

-initworker=*number*
The initial number of Worker threads.

-maxworker=*number*
The maximum number of Worker threads. If this value is set to -1, no limits are applied to the number of ConnectionManager threads.

-trace=[on|off]
Enables or disables extra tracing messages. You can also specify a file for the messages, for example:

```
jgate -trace > trc001
```

-time=[on|off]
Enables or disables timing information in messages.

-noinput=[on|off]
Enables or disables the reading of input from the console.

Operation

-nonames=[on|off]
Enables or disables the lookup of TCP/IP host names for printing the connect and disconnect messages. This can improve performance, and it allows the CICS Transaction Gateway for OS/390 to be used on systems that have no domain name server.

To override the startup defaults, type: `JGate` at the command prompt, followed by the startup options you require, and press Enter.

You see the startup message:

```
CCL6501I: Starting the CICS Transaction Gateway with user specified values.
```

This is followed by two lines showing the values that are being used, for example:

```
CCL6502I: [ Port = 2006 , Initial Connections = 10 , Maximum Connections = 100 ,
CCL6502I:   Initial Workers = 10 , Maximum Workers = 100 ]
```

To get help for the startup options, enter: `JGate ?`

Starting with JCL

This is a sample of the JCL you can use to start the CICS Transaction Gateway for OS/390 as a batch job:

```
//DFHJGATE JOB (Accounting info),CLASS=A,USER=user,PASSWORD=passwd,
//         MSGCLASS=H
//OEEXCI   EXEC PGM=BPXBATCH,
//           PARM='SH /ctg/bin/JGate -noinput',
//           REGION=8M
//STDIN  DD PATH='/dev/null',
//          PATHOPTS=(ORDONLY)
//STDOUT DD PATH='/jgateo.log',PATHOPTS=(OWRONLY,OCREAT),
//          PATHMODE=SIRWXU
//STDERR DD PATH='/jgatee.log',PATHOPTS=(OWRONLY,OCREAT),
//          PATHMODE=SIRWXU
//STDENV DD *
DFHJVPIPE=JAVAGAT1
DFHJVSYSTEM_00=IJKLMNOP-Primary CICS server
DFHJVSYSTEM_01=OTHER-Some other CICS system
/*
//
```

In this example:

- BPXBATCH is the MVS program that executes an OS/390 UNIX System Services script as a batch job. The PARM field specifies that the shell (SH) is to execute the specified `JGate` command with the `-noinput` option. The path assumes that the top-level /ctg directory is directly accessible from the HFS root.
- STDENV is a data set in which you can set the values of any environment variables. In the example, the DFHJVPIPE, DFHJVSYSTEM_00, and DFHJVSYSTEM_01 environment variables are set (see “Environment variables” on page 131 for more information). The values to be given to the environment variables must not be enclosed in quotes. If a line has a sequence numbers, the sequence number is interpreted as part of the value of the environment variable. Values specified in JCL are overridden by values specified in the JGate script (see “Customizing the JGate script” on page 136).

Stopping the CICS Transaction Gateway for OS/390

If you started the CICS Transaction Gateway for OS/390 from the command line, and if you did not specify the `-noinput` parameter, you can stop the Gateway by typing `Q` and pressing Enter in the Gateway console session.

If you have used the `-noinput` parameter, or if you used JCL to start the CICS Transaction Gateway for OS/390, you must stop the Gateway process by using the JES CANCEL command.

Chapter 26. CICS Transaction Gateway for OS/390 programming overview

This chapter provides an introduction to Java programming with the CICS Transaction Gateway for OS/390. Detailed information on programming is given in “Appendix I. Programming reference information for the CICS Transaction Gateway for OS/390” on page 239.

Programming interface

The CICS Transaction Gateway for OS/390 provides the following classes and interfaces, which make up its public programming interface.

- **Java-client program classes**
  - com.ibm.ctg.client.JavaGateway
  - com.ibm.ctg.client.ECIRequest
  - com.ibm.ctg.client.EPIRequest
  - com.ibm.ctg.client.CicsCpRequest
  - com.ibm.ctg.client.Callbackable (interface)
  - com.ibm.ctg.client.GatewayRequest
- **Interface definitions for writing Gateway security classes**
  - com.ibm.ctg.security.ClientSecurity
  - com.ibm.ctg.security.ServerSecurity

Writing Java-client programs

At the simplest level, the flow of program control needed to write a simple CICS Transaction Gateway for OS/390 Java-client program is as follows:

1. The Java program creates and opens an instance of a com.ibm.ctg.client.JavaGateway object.
   - The default JavaGateway constructor creates a blank JavaGateway object. You **must** then set the correct properties in this object using the relevant set.. methods. The JavaGateway is then opened by calling the open method.
   - Two other JavaGateway constructors exist that simplify the creation of a JavaGateway by setting the relevant properties and implicitly calling the open method for you. On return from a successful call to one of these constructors, the resultant JavaGateway is open and connected to the requested CICS Transaction Gateway for OS/390.
2. The Java program creates an instance of the com.ibm.ctg.client.ECIRequest object containing the request that it wishes to make.
3. The Java program then flows the request to the CICS Transaction Gateway for OS/390 using the flow method of the JavaGateway object.
4. The Java program checks the return code of the flow operation to see whether the request was successful.
5. The program continues to create request objects and flow them through the JavaGateway object, as appropriate.
6. The Java program then closes the JavaGateway object.

The CICS Transaction Gateway for OS/390 also provides a sample program TestECI, to illustrate the use of these classes.

TestECI

TestECI is a sample program that allows you to test the functionality of the CICS Transaction Gateway for OS/390. With TestECI you can connect to a Gateway and then send one or more ECI requests to a CICS server. If you specify more than one CICS program on the server, all the programs are run as one extended Logical Unit of Work (LUW). You can run TestECI either as an application, or as an applet.

The source for TestECI is provided in the directory:

```
/ctg/samples/java/com/ibm/ctg/test
```

Running TestECI as an application

When running TestECI as an application, parameters are passed in via the command line and output appears in the console.

The syntax is:

```
java com.ibm.ctg.test.TestECI [jgate=jgate_URL]
                              [jgateport=jgate_port]
                              [clientsecurity=client_security_class]
                              [serversecurity=server_security_class]
                              [server=cics_server]
                              [userid=cics_userid]
                              [password=cics_password]
                              [prog<0..9>=prog_name]
                              [commarea=comm_area]
                              [commarealength=comm_area_length]
                              [status]
                              [trace]
```

Where :

- *jgate_URL* is the URL of the Gateway to connect to.
- *jgate_port* is the TCP/IP port to connect to on jgate_server, if it was not specified as part of the *jgate_URL*.
- *client_security_class* is the name of the class to use to provide client-side security
- *server_security_class* is the name of the class to use to provide server-side security
- *cics_server* is the name of the CICS server to receive ECI requests.
- *cics_userid* and *cics_password* are the userid and password.
- *prog_name* is the name of a CICS server program. You can specify up to ten program names.
- *comm_area* is the initial value of the COMMAREA, if any
- *comm_area_length* is the length of the COMMAREA to send to each CICS server program
- `status` causes the program to query the status of all the known CICS servers.
- `trace` causes tracing information to be produced.

For example:

```
java com.ibm.ctg.test.TestECI jgate=myjgate.here.com server=mycics
commarea="Hello World" prog0=testprog prog1=testprog2 status
```

Running TestECI as an applet

When running TestECI as an applet you pass in parameters via <param> tags within the <applet> tag. Output appears in a text area on the browser running the applet.

The parameters are the same as those used when running TestECI as an application (see “Running TestECI as an application” on page 150).

```
<applet code="TestECI" align="baseline" width="128 height="128" .....>

<param name="jgate" value="jgate_URL">
<param name="jgateport" value="jgate_port">
<param name="clientsecurity" value="client_security_class">
<param name="serversecurity" value="server_security_class">
<param name="server" value="cics_server">
<param name="userid" value="cics_userid">
<param name="password" value="cics_password">
<param name="progn" value=prog_name">
<param name="commarea" value="comm_area">
<param name="commarealength" value="comm_area_length">
<param name="status" value="yes">
<param name="trace" value="yes">
</applet>
```

If *jgate_URL* is not specified, the browser connects to the applet host.

Sample HTML to invoke TestECI as an applet is provided in testeci.html, which is located with the TestECI source.

Making ECI calls

The CICS Java classes allow you to create **ECIRequest** objects that represent calls to the ECI.

Program link calls

You can create **ECIRequest** objects that represent requests for synchronous and asynchronous program link calls (ECI_SYNC and ECI_ASYNC call types).

The following restrictions apply to the CICS Transaction Gateway for OS/390:

- The length of the communication area passed to the ECI must not exceed 32 659 bytes.
- You must specify a callback routing with an asynchronous call. Reply solicitation calls are not supported.
- You must specify ECI_NO_EXTEND for **eci_extend_mode**, as extended units of work are not supported.
- You can specify any values for **eci_password**, **eci_luw_token**, and **eci_message_qualifier**, but they are ignored.

Status information calls

The ECI status information calls (ECI_STATE_SYNC and ECI_STATE_ASYNC call types) return their results in a CICS communication area. Since the contents of the

communication area are platform dependent, the **ECIRequest** class provides help with interpreting the results of a status information call:

- Public variables to hold the results of a status information call in a platform-independent manner
- Two call types (ECI_STATE_SYNC_JAVA and ECI_STATE_ASYNC_JAVA) that return the results of a status information call in the public variables rather than in a communication area
- Methods to interpret the contents of the public variables as strings for display

The CICS Transaction Gateway for OS/390 supports only the ECI_STATE_IMMEDIATE value for the extend mode in status information calls.

Reply solicitation calls

You can create **ECIRequest** objects that represent reply solicitation calls.

The CICS Transaction Gateway for OS/390 does not support reply solicitation calls.

CICS security considerations

The CICS Transaction Gateway for OS/390 region is an EXCI user, so the following security considerations apply:

- If the user ID of the CICS Transaction Gateway for OS/390 address space is *the same as* the user ID of the CICS address space:
  - LINK security checking is not performed.
  - Surrogate user checking is performed, so the user ID of the CICS Transaction Gateway for OS/390 address space must be authorized to use the user ID passed on the ECI call.
- If the user ID of the CICS Transaction Gateway for OS/390 address space is *different from* the user ID of the CICS address space, LINK security checking is performed according to the setting of ATTACHSEC for the connection. The user ID passed on the ECI call is checked as valid.

ECI return codes and server errors

This section describes how the return codes from the EXCI and from the CICS business logic interface are returned to the user of the **ECIRequest** object.

Table 2 shows how EXCI return codes map to ECI return codes. The EXCI return codes are documented in the *CICS External Interfaces Guide*.

Table 2. EXCI return codes and ECI return codes

EXCI return codes	ECI return codes
201, 203	ECI_ERR_NO_CICS
202	ECI_ERR_RESOURCE_SHORTAGE
401, 402, 403, 404, 410, 411, 412, 413, 418, 419, 421	ECI_ERR_SYSTEM_ERROR
422	ECI_ERR_TRANSACTION_ABEND
423	ECI_ERR_SECURITY_ERROR
601, 602, 603, 604, 605, 606, 607, 608, 621, 622, 623, 627, 628	ECI_ERR_SYSTEM_ERROR

Table 2. EXCI return codes and ECI return codes (continued)

EXCI return codes	ECI return codes
609	ECI_ERR_SECURITY_ERROR
624	ECI_ERR_REQUEST_TIMEOUT

Table 3 shows how CICS business logic interface return codes map to ECI return codes. The CICS business logic interface return codes are documented in the *CICS External Interfaces Guide*.

Table 3. CICS business logic interface return codes and ECI return codes

CICS business logic interface return codes	ECI return codes
400	ECI_INVALID_DATA_AREA
403	ECI_SECURITY_ERROR
404	ECI_TRANSACTION_ABEND
500, 501	ECI_ERR_SYSTEM_ERROR

Making EPI calls

The CICS Java classes allow you to create **EPIRequest** objects that represent calls to the EPI. You set the public variable `Call_Type` in an **EPIRequest** object to specify which EPI call you wish to make. The results of the EPI call are returned in the object after you use the `flow` method of the **JavaGateway** object.

However, as you might expect, the CICS Transaction Gateway for OS/390 rejects attempts to flow an **EPIRequest** object. The return code EPI_ERR_FAILED is returned. You should not use **EPI_GetSysError** to attempt to get more information. If you wish to run transactions in the manner of the EPI, you should use the ECI and set up a request for DFHWBTTA.

Chapter 27. CICS Transaction Gateway for OS/390 problem determination

Problem determination is not to be confused with problem solving, although while investigating a problem you may find enough information to solve the problem. Examples of the types of problem that can arise are:

- End-user errors
- Programming errors
- Configuration errors.

Preliminary checks

Before investigating the problem, it is worth checking to see whether there is an obvious cause:

- Has the system run successfully before?
- Have you made any changes to the configuration of the system or added new features or programs?
- If you have migrated from using the old CICS Transaction Gateway for OS/390, are you sure that the configuration file (Gateway.properties), and JGate script were migrated successfully?
- Are your environment variables, such as CLASSPATH and LIBPATH set correctly? (See “Environment variables” on page 131.)
- Are there any messages explaining the failure?
- Can the failure be reproduced?

Problems running sample applets using the JDK AppletViewer

When using AppletViewer to run any of the CICS Transaction Gateway sample applets from your local filesystem, the message CCL6664E `Unable to load relevant class to support the` *xyz* `protocol` may be displayed.

To resolve this problem, select **Applet** from the AppletViewer menu bar; then select **Properties**. Then set both **Network Access** and **Class Access** to `Unrestricted`.

Conflicts with default ports

The CICS Transaction Gateway uses default ports for its supported protocols. These may conflict with ports already in use, and if this is the case, one or more protocols fail to start successfully. You can change port number in the Gateway.properties file, see “Configuring the Gateway.properties file” on page 132.

What to do next

If you think the problem is in the CICS Transaction Gateway for OS/390, you need to collect as much information as possible and contact your support organization.

If you started the Gateway without **trace** enabled, you need to stop the Gateway, restart it with the **trace** option, and recreate the problem.

If you suspect the problem is elsewhere in the network, you should follow the problem determination procedures provided with those other products. See the *CICS Problem Determination Guide*.

Program support

Different levels of program support are available and you should check what level you have before contacting IBM. Warranty and support information is provided in the *License* documents you get with the product; some products also include a **Service and Support** card.

Messages

The CICS Transaction Gateway for OS/390 messages have a *CCL* prefix; which has traditionally been used for CICS Clients.

For a list of the messages generated by the CICS Transaction Gateway for OS/390, see “Chapter 28. Messages” on page 161

Sources of information

You can get problem determination information from the following sources:

- The standard output (stdout) and standard error (stderr) files:
  - Standard output contains a log of CICS Transaction Gateway messages, and any messages returned to the EXCI from CICS.
  - Standard error contains the messages in standard output, together with error messages from the Java virtual machine. The Java virtual machine messages are documented in the Java development toolkit (JDK).
- CICS Transaction Gateway for OS/390 trace: This shows the activity in the CICS Transaction Gateway for OS/390 when it is handling a request from a Web browser. This trace is written as to the EXCI trace, and the trace points are listed in Table 4 on page 157.
- EXCI trace: This shows the EXCI activity during the call to the CICS business logic interface. The CICS Transaction Gateway for OS/390 trace is part of the EXCI trace. For details of EXCI tracing, see the *CICS External Interfaces Guide*
- CICS trace: This shows the progress of the request through:
  - The CICS mirror transaction that handles the EXCI request
  - The CICS business logic interface, which does the data conversion
  - The called CICS program.

  You should examine the AP and WB trace points in the CICS trace.

To route messages and trace information to a file, specify:

```
jgate -trace > trc001
```

when starting the Gateway, where `trc001` is the name of your output file.

Tracing in the CICS Transaction Gateway for OS/390

The CICS Transaction Gateway for OS/390 writes trace entries to a buffer in its address space. The trace entries are in the CICS trace EXCI format, so the trace entries in a dump can be printed using standard CICS utilities. You can use the following operating procedure from the SDFS system log command line:

1. Use the command

   ```
   /D OMVS,A=ALL
   ```

   to display OMVS tasks.
2. Find the CICS Transaction Gateway for OS/390 task, and note the ASID.
3. Enter the DUMP command with a suitable comment. For example:

   ```
   /DUMP COMM=(JGATE DUMP)
   ```
4. Reply to the message with the ASID as follows:

   ```
   /R nn,ASID=aa,END
   ```

   where *nn* is the message number for the reply, and *aa* is the ASID.

Table 4 shows the trace points written to the EXCI trace by the CICS Transaction Gateway for OS/390.

Table 4. EXCI trace points for the CICS Transaction Gateway for OS/390

Point ID	Module	Lvl	Type	Data
8000	JVDLL	EX 1	ECI parameters passed	**1** Thread name **2** `Call_Type` **3** `Extend_Mode` **4** `Luw_Token` **5** `Commarea_Length` **6** `Cics_Rc` **7** `Message_Qualifier`
8001	JVDLL	Exc	GetStringPlatform error	**1** Return code **2** Data area **3** Length
8002	JVDLL	Exc	GetStringPlatformLength error	**1** Return code
8003	JVDLL	EX 1	Converted parameter	**1** Thread name **2** Parameter name **3** Parameter value

Table 4. EXCI trace points for the CICS Transaction Gateway for OS/390 (continued)

Point ID	Module	Lvl	Type	Data
8004	JVDLL	EX 1	Inbound COMMAREA	**1** Thread name **2** Communication area length **3** Communication area address **4** 250 bytes of data
8005	JVDLL	EX 1	DFHWBA1 parameters	**1** Thread name **2** Converter program name **3** Server program name **4** Data offset **5** Communication area length **6** Communication area address
8006	JVDLL	EX 1	Outbound COMMAREA	**1** Thread name **2** Communication area length **3** Communication area address **4** 250 bytes of data
8007	JVDLL	EX 1	ECI parameters output	**1** Thread name **2** `Call_Type` **3** `Extend_Mode` **4** `Luw_Token` **5** `Commarea_Length` **6** `Cics_Rc` **7** `Message_Qualifier`

Table 4. EXCI trace points for the CICS Transaction Gateway for OS/390 (continued)

Point ID	Module	Lvl	Type	Data
8010	JVDLL	Exc	Error response received	**1** Function number **2** EXCI response **3** EXCI reason **4** EXCI subreason field-1 **5** EXCI subreason field-2 **6** `Cics_Rc`
8011	JVDLL	Exc	DPL_REQUEST error	**1** RESP **2** RESP2 **3** ABCODE **4** `Cics_Rc`
8012	JVDLL	Exc	WBA1 parameters allocation error	**1** malloc length **2** `Cics_Rc`
8013	JVDLL	Exc	Invalid call type	**1** Thread name **2** `Call_Type` **3** `Cics_Rc`
8014	JVDLL	Exc	Invalid COMMAREA length	**1** Thread name **2** `Commarea_Length` **3** Size of `Commarea` **4** `Cics_Rc`

Chapter 28. Messages

The CICS Transaction Gateway messages have a CCL prefix. Message suffixes are are follows:

- I — information; no user action.
- W— warning; no user action.
- E — error; user action as specified in the message description.

CCL6500I Starting the CICS Gateway for Java with default values.

Explanation: The user entered a start command with no options.

System Action: The gateway is started with default options.

User Response: None.

CCL6501I Starting the CICS Gateway for Java with user specified values.

Explanation: The user entered a start command with user-specified options.

System Action: The gateway is started with the user-specified options.

User Response: None.

CCL6502I ...

Explanation: The options in effect are as listed in the message text.

System Action: None.

User Response: None.

CCL6503I Starting with tracing messages on.

Explanation: The user entered a start command with the -trace option.

System Action: The gateway is started with the tracing messages on.

User Response: None.

CCL6505I Successfully created the initial connection and worker threads.

Explanation: The intial number of connection manager and worker threads have been created.

System Action: None.

User Response: None.

CCL6506I Client connected *cmthread* on socket *ipaddress*.

Explanation: A browser has connected from the IP address *ipaddress*, and the connection is being managed by the connection manager thread *cmthread*.

System Action: None.

User Response: None.

CCL6507I Client disconnected *cmthread* on socket *ipaddress*.

Explanation: A browser has disconnected from the IP address *ipaddress*

System Action: The connection manager thread *cmthread* is now available for reuse.

User Response: None.

CCL6508I Type Q to stop the CICS Gateway for Java.

Explanation: The gateway can be stopped by the Q console command.

System Action: None.

User Response: None.

CCL6509I The CICS Gateway for Java is stopping.

Explanation: A command to stop the gateway has been entered.

System Action: In-progress requests will be cleaned up.

User Response: None.

CCL6511I ThreadManger created *numbertype*.

Explanation: The thread manager created *number* threads of type *type*.

System Action: None.

User Response: None.

CCL6512I Wating to receive a request *cmthread*.

Explanation: The connection manager thread *cmthread* is waiting to receive a request.

System Action: None.

User Response: None.

CCL6513I About to receive remainder of request *requesttype*.

Explanation: A connection manager thread has read a request header of type *requesttype*, and is waiting for the rest of the request.

System Action: None.

User Response: None.

CCL6514I Accepting work request. *wthread* *cmthread* to socket *ipaddress*.

Explanation: The worker thread *wthread* is accepting work from connection manager thread *cmthread* which is working with a browser at IP address *ipaddress*.

System Action: None.

User Response: None.

CCL6515I Finished work request *wthread*.

Explanation: The worker thread *wthread* has finished its work and is now available for reuse.

System Action: None.

User Response: None.

CCL6516W Outstanding work still in progress.

Explanation: A browser has disconnected while there is still work in progress for it.

System Action: The work is completed, but the results are discarded.

User Response: None.

CCL6517I All outstanding work has been finished.

Explanation: Work for a disconnected browser has now finished and the results discarded.

System Action: None.

User Response: None.

CCL6518W Unfinished extended requests were in progress *number*.

Explanation: A browser disconnected while *number* extended requests were in progress.

System Action: The requests will be cleaned up.

User Response: None.

CCL6519I Attempts have been made to clean up all unfinished extended requests.

Explanation: Extended requests abandoned by a browser have been cleaned up.

System Action: None.

User Response: None.

CCL6520I This is the first request of an extended request sequence.

Explanation: A request that starts a series of extended requests is now being processed.

System Action: None.

User Response: None.

CCL6521I This request is part of an extended request sequence.

Explanation: A request that is one of a series of extended requests is now being processed.

System Action: None.

User Response: None.

CCL6522I This is the final request of an extended request sequence.

Explanation: A request that ends a series of extended requests is now being processed.

System Action: None.

User Response: None.

CCL6523I About to execute work request *wthread*.

Explanation: The worker thread *wthread* is about to start a work request.

System Action: None.

User Response: None.

CCL6560E Unable to listen on requested port.

Explanation: The gateway cannot listen on the port specified.

System Action: The gateway is not started.

User Response: If there is another gateway already using the port, start this gateway with another port.

CCL6561E Unable to create requested connection and worker threads.

Explanation: The gateway cannot create the requested initial numbers of connection manager and worker threads.

System Action: The gateway is not started.

User Response: Try starting the gateway with fewer threads.

CCL6562E Error whilst accepting a connection. Connection closed.

Explanation: An error occurred when a new browser tried to connect to the gateway.

System Action: The connection is refused.

User Response: Retry.

Part 6. CORBA client support

This part of the book describes CICS support for inbound IIOP requests for CICS JAVA applications. It covers the following topics:

IIOP inbound to Java

Chapter 29. IIOP inbound to Java®

The Internet Inter-ORB protocol (IIOP), is an industry standard that defines formats and protocols to provide client/server semantics for distributed object-oriented application programs in a TCP/IP network. It is part of the Common Object Request Broker Architecture (CORBA) specification.

CICS Transaction Server for OS/390 Release 3 provides support for inbound requests to Java application programs, using the IIOP protocol. Execution of Java server programs requires the VisualAge for Java, Enterprise ToolKit for OS/390. For information about building Java applications to run in CICS, and the use of the CICS Java classes, see the *CICS Application Programming Guide*.

A subset of CORBA services is provided, suitable for distributed objects that have evolved from existing CICS applications and therefore have the following characteristics:

- State by virtue of their explicit use of CICS resources, rather than state that is managed by the Object Request Broker (ORB). State is initialized at the start of each method call and referenced by explicit method parameters.
- Transaction and security contexts managed by CICS facilities, so these CORBA services are not provided.
- CICS services used to reference distributed applications, so outbound object references are not supported.
- Applications and their interfaces predefined, so the Dynamic Skeleton Interface (DSI) is not supported.

With any distributed application, the client and server need basic information to be able to communicate, such as information about the available operations the client can request, and the arguments to the operations. This information is provided by an interface that you define using the Object Management Group (OMG) Interface Definition Language (IDL) to code a set of **interface definitions**.

Each method call is implemented as a CICS transaction.

Workload balancing of requests is implemented at three levels:

TCP/IP port sharing
TCP/IP port sharing is provided by the eNetwork Communications Server in OS/390 Version 2 Release 5 or later. See *TCP/IP for MVS: Customization and Administration Guide* and *OS/390 eNetwork Communications Server: IP Configuration Guide* for further information.

Dynamic Domain Name Server (DNS) registration for TCP/IP
Balances IP connections and workload in a Sysplex domain. The Initial Interoperable Object reference (IOR) to the CICSplex contains a generic host name and port number. With dynamic DNS, multiple CICS systems are started to listen for IIOP requests on the same port (using Virtual IP addresses), and the host name in the initial IOR is resolved to an IP address by MVS DNS and Workload Management (WLM) services.

Connection Optimization in a Sysplex Domain is described in the *OS/390 TCP/IP Update Guide GC31-8553*.

CICS Dynamic Linkage
Balances method call invocations across CICS regions. The dynamic selection of the target is provided by CICS services, selecting the least loaded or most efficient application region.

The following diagram shows the two levels of workload balancing:

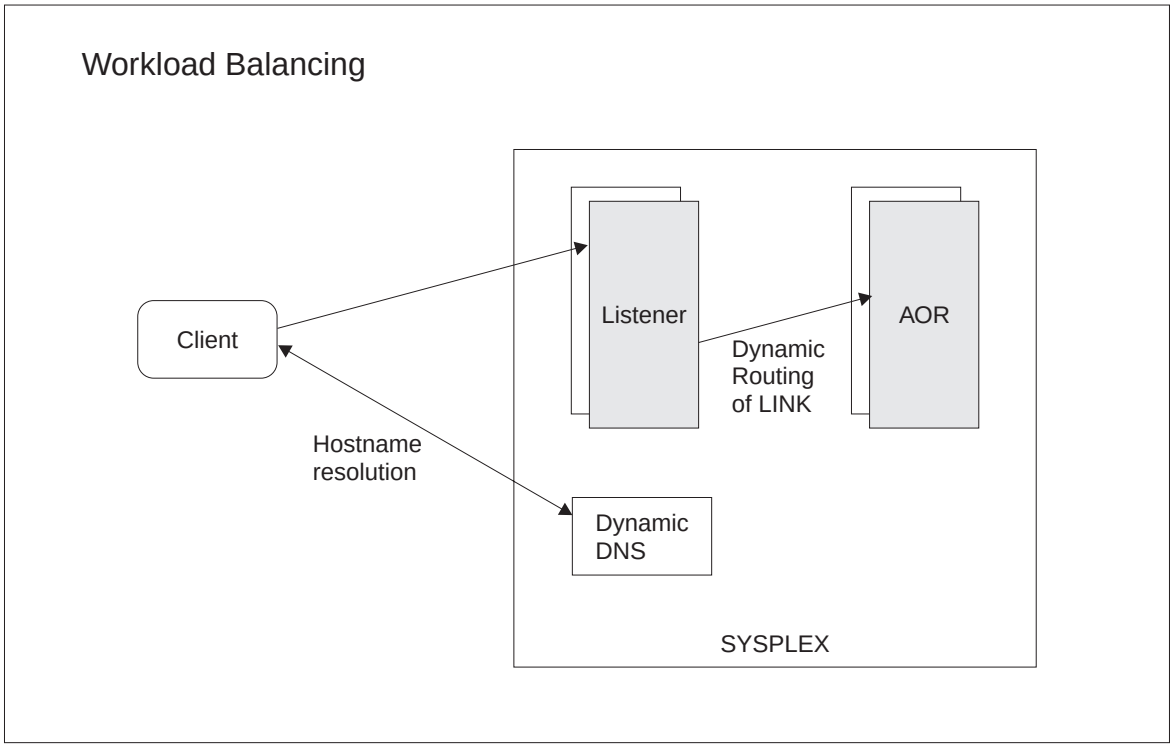


Figure 23. Workload Balancing using DNS

Terminology

the following terms are used throughout this part of the book:

OMG The Object Management Group. The consortium of software organizations that has defined the CORBA architecture.

CORBA
The Common Object Request Broker Architecture. An architecture and a specification for distributed object-oriented computing.

ORB The Object Request Broker. A CORBA system component that acts as an intermediary between the client and server applications. Both client and server platforms require an ORB; each is tailored for a specific environment, but support common CORBA protocols and IDL.

IIOP The Internet Inter-Orb Protocol. An industry standard that defines formats and protocols to provide client/server semantics for distributed object-oriented applications in a TCP/IP network. It is part of the CORBA architecture.

IDL Interface Definition Language. A definition language that is used in CORBA to describe the characteristics and behavior of a kind of object, including the operations that can be performed on it.

Module
This maps to a Java **package**.

Interface
Describes the characteristics and behavior of a kind of object, including the operations that can be performed on those objects. This maps to a **class**. In CORBA terminology, the client request specifies, in IDL, an interface that defines the server object.

Operation
An action that can be performed on an object. This maps to a **method**. In CORBA terminology, the client requests an operation, defined in IDL, that is mapped to a method on the server object.

IOR Interoperable Object Reference. In a distributed environment this provides enough information to locate the server and the object.

Stub or proxy
This is generated by the client IDL compiler. It is used by the ORB to convert a local object reference to an IOR, and invoke translation of object datatypes from/to the IIOP message syntax.

Skeleton
This is generated by the server IDL compiler. It is used by the ORB to parse the message into a method call on a local (to the server) object.

Execution flow

The following diagram shows the execution flow of an incoming request:

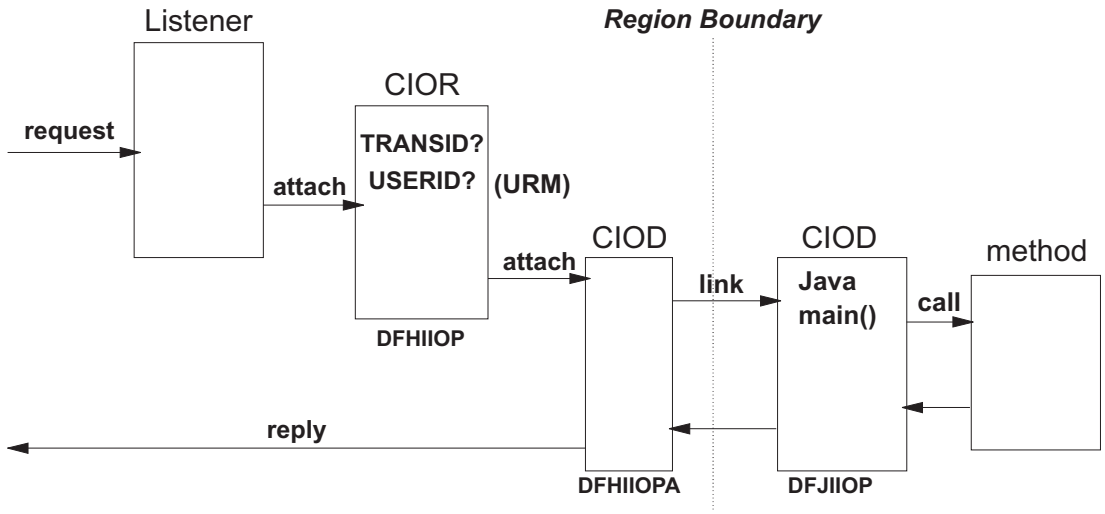


Figure 24. IIOP request execution flow

Listener
The CICS TCP/IP listener monitors specified ports for inbound requests. IIOP ports are specified by defining and installing TCPIPSERVICE resources. See “Chapter 31. Processing the IIOP request” on page 175 for more information about the TCPIPSERVICE resource.

The TCPIPSERVICE resource definition also controls dynamic DNS load balancing.The selected Listener receives the incoming request and starts the

transaction specified in the TCPIPSERVICE definition for that port. For IIOP services, this transaction should be **CIOR**, executing the CICS receiver program **DFHIIOP**.

Establishing execution characteristics

DFHIIOP retrieves the incoming request and matches its interface and operation (class and method) against templates defined by REQUESTMODEL resource definitions. The selected REQUESTMODEL provides the name of the CICS transaction under which the method will run. If no match is found, the default transaction **CIOD** is used. You can define your own transaction, with any name, to provide the transaction execution characteristics, but the program name must be **DFHIIOPA**.

DFHIIOP then calls a User Replaceable Module (URM) to supply a USERID, and attaches the requested CICS transaction, passing it the inbound IIOP request data. You can define the name of the URM in the TCPIPSERVICE resource definition for the IIOP port. If no name is specified, the default DFHXOPUS will be called.

DFHIIOP then attaches the requested transaction (default CIOD) to run the DFHIIOPA program with the requested USERID.

ORB function

DFHIIOPA links to DFJIIOP to handle the IIOP request. This linkage can exploit CICS dynamic routing services to provide load balancing within the CICSplex. Note that it is DFJIIOP that is routed, not the method. To do this, you need to make copies of the supplied default transactions (CIOD and CIOF), changing the PROGRAM name to DFHMIRS, and install them in the AOR.

DFJIIOP analyzes the contents of the IIOP request (in the passed COMMAREA or TS queue) and then:

- Instantiates the target object
- Demarshals the input parameters
- Invokes the requested method on the target object. This can access CICS resources and link to other CICS application programs using the CICS Java (JCICS) classes. (See the *CICS Application Programming Guide* for information about the CICS Java API).
- Marshals the reply and returns it to DFHIIOPA for transmission back to the sender of the IIOP request.

A client ORB may also generate IIOP LocateRequest messages, which are handled in a similar manner.

CORBA Services support

Name Server support

Name server support is not implemented in CICS. A stringified reference to the CosLifeCycle::GenericFactory implemented in the server can be written to a file using the GenFacIOR utility class, and you must ensure that this stringified reference is available to clients.

Security support

Security support is provided by CICS rather than a CORBA IIOP mechanism. All IIOP requests to CICS will run under a default USERID unless you provide a

user replaceable module to generate a USERID for each request. See “Obtaining a CICS USERID” on page 178 for more information about the IIOP user replaceable module.

Lifecycle support
Only the CORBA GenericFactory interface is supported, implemented in program DFJGFAC. Unless overridden, GenericFactory requests will run under the CIOF transaction, as shown in “Supplied REQUESTMODEL definitions” on page 177.

Externalization
The externalization service is not supported.

Persistence
The persistence service is not supported.

Concurrency
The concurrency service is not supported.

Interface Repository Framework
The interface repository framework service is not supported.

Location service
The location service is not used. All object references refer either to a specific server, or if workload balancing is in use, to a server group.

Chapter 30. Requirements for IIOP applications

This chapter describes the libraries and files that you will need to develop and run IIOP applications, and the CICS resource definitions required.

Environment

To build a CICS Java server program, you will require the following environment:

- An MVS/ESA system configured with Full Function OS/390 UNIX System Services (previously known as OpenEdition®)
- CICS Transaction Server for OS/390 Release 3 with Language Environment (LE) active
- A Java compiler such as javac, installed on OS/390 UNIX System Services, or on a workstation that can connect to the OS/390 UNIX System Services environment to transfer data, or VisualAge for Java installed on a workstation
- The VisualAge for Java, Enterprise ToolKit for OS/390 installed on ESA

CICS parameters

You should review the following parameter settings in the CICS system intialization table:

EDSALIM
Memory requirements to run Java programs using ET/390 are higher than for conventional programs. You should set the system initialization parameter EDSALIM to a high value (such as 100MB) when starting CICS, otherwise a Short-on-Storage condition may occur. Note that this must be set by SIT override, not using CEMT SET commands.

MXT
CICS requires two transactions to process each request, so you should increase the maximum task limit (MAXTASKS) by setting the MXT parameter in the CICS system initialization table appropriately. The CIOR transaction should be defined in a TRANCLASS whose MAXACTIVE task value is less than half the MXT value.

.jar files

The following CICS supplied files are required in your CLASSPATH. They are stored in the OS/390 UNIX System Services HFS in a directory **$CICS_HOME/classes** during CICS installation:

dfjcidl.jar
The CICS IDL compiler to be used in building the IIOP server application.

dfjcorb.jar
The CICS ORB classes, required to build the IIOP server application This also contains the GenFacIOR utility that you need to build your client program.

dfjcics.jar
The JCICS API classes, required for compilation of a Java server program that uses JCICS to access CICS services.

$CICS_HOME is an environment variable defining the installation directory prefix:

```
/usr/lpp/cicsts/<username>
```

Where **username** is a name you can choose during the installation of CICS, defaulting to `cicsts13`.

CICS libraries

The following CICS PDSE libraries are required in the CICS DFHRPL library concatenation at run time.

IIOP and JCICS

The MVS PDSE library **SDFJLOAD** (or **SDFJLOD1**) is required. These libraries are built during CICS installation. SDFJLOAD is maintained at a level compatible with the current release of the VisualAge for Java, Enterprise ToolKit for OS/390 (ET/390), and SDFJLOD1 is maintained at a level compatible with Release 1. You will only require one of these libraries and should choose the one that is compatible with the release of ET/390 that you are using.

Program libraries

A PDSE library is required to hold the CICS Java server program objects that have been bound by ET/390.

Note: PDSE libraries are similar to PDS libraries. They contain directories and members, but allow long-name aliases for the 8-byte Primary Member names. You can use them either for data, or for programs (but not a mix of both), and combine both PDS and PDSE libraries in the same concatenation.

Resource definitions

CICS resources, such as PROGRAMS and TRANSACTIONS must all be defined to CICS. Resource definitions for the supplied IIOP components are provided in group DFHIIOP, which is included in GRPLIST. You should not need to change these definitions, but you must provide resource definitions for your own CICS programs. See the *CICS Resource Definition Guide* for information about CICS resource definition.

Chapter 31. Processing the IIOP request

The IIOP request is received by the CICS TCP/IP Listener and the requested CICS transaction is started. This part of the book tells you how to register an IIOP service with the CICS TCP/IP Listener and what you need to do to establish the execution environment for the CICS server ORB function. It covers the following topics:

Registering with the CICS TCP/IP Listener

The CICS TCP/IP Listener receives incoming IIOP requests from the ports that you have registered by defining and installing TCPIPSERVICE resources.

The TCPIPSERVICE definition allows you to specify:

- the port or IP address on which CICS will listen for incoming requests
- the CICS transaction to start when a request arrives. For an IIOP service, this should be set to CIOR, as shown in the example
- the level of secure sockets layer (SSL) authentication to be used
- the name of the user replaceable module (URM) to be called. For IIOP, this defaults to DFHXOPUS

See the *CICS Resource Definition Guide* for full details of the TCPIPSERVICE resource definition.

Dynamic Name Server

To select the dynamic name server (DNS), you need to use a TCPIPSERVICE whose name begins with '**D**'. The Listener registers with MVS workload management services (WLM) using the following values:

- The TCPIPSERVICE TRANSID is passed to MVS as the **Group** name. If the TCPIPSERVICE name is of the form `Dxxx.yyyy` then CICS uses `yyyy` as a prefix to TRANSID name. For example, a TCPIPSERVICE of DEV.CICS with a TRANSID of CIOR generates a group name of CICSCIOR.
- The APPLID specified in the system initialization table (SIT) is passed to MVS as the **Server**. If the APPLID=(gname,sname) format is used in the SIT, then **sname** is the value passed to MVS.

Notes:

1. Both the client and the CICS server must use the same TCP/IP **nameserver**
2. The **nameserver** must be able to perform a reverse look-up, that is, it must be able to translate the IP address of the server into a full hostname

TCPIPSERVICE example

The following sample TCPIPSERVICE resource definition is supplied. You can modify it to suit your requirements:

```
DEFINE TCPIPSERVICE(IIOPNSSL) GROUP(DFH$SOT)
DESCRIPTION(IIOP TCPIPSERVICE with no SSL)
BACKLOG(5)
PORTNUMBER(535)
TRANSACTION(CIOR)
SSL(NO)
STATUS(OPEN)
```

Obtaining a CICS TRANSID

When a request is received, the CIOR transaction is initiated and the CICS provided DFHIIOP program receives control.

The incoming message has an IIOP standard format, defined by the CORBA architecture. DFHIIOP compares the message with REQUESTMODEL resource definitions that you have previously defined and installed, and selects the closest match. The selected REQUESTMODEL provides the name of the CICS transaction under which the method will run. If no match is found, this defaults to CIOD.

The matching process compares the **Module** name, **Interface** and **Operation** fields contained within the IIOP message, against those defined in each installed REQUESTMODEL, until the closest match is found. The following parameters can be specified on a REQUESTMODEL resource definition:

REQUESTMODEL
The name of the REQUESTMODEL resource being defined.

DESCRIPTION
A description of the REQUESTMODEL resource being defined.

OMGMODULE
Defines a pattern which may match the qualified module name (coded in OMG IDL) that defines the name scope of the interface and operation whose implementation is to be executed. Each component of the module, and the interface and operation names, are CORBA identifiers made up from the following characters:

- Alphabetic, including accented characters.
- Numeric digits
- Underscore

The first character must be alphabetic. Case is not significant, however, mixed case is enabled for ease of use when referencing implementations in mixed case, such as Java. Module components must be separated by a double colon (::). This is the IDL equivalent of the Java *package* name.

OMGINTERFACE
Defines a pattern that may match the IDL interface name. This is the IDL equivalent of the Java *class* name.

OMGOPERATION
Defines a pattern that may match the IDL operation name. This is the IDL equivalent of the Java *method* name.

TRANSID
Defines the 4-character name of the CICS transaction to be executed when a request matching the REQUESTMODEL is received. This transaction must be

defined to CICS with a TRANSACTION resource definition with the PROGRAM parameter set to DFHIIOPA. You can base your transaction definition on the supplied CIOF definition.

See the *CICS Resource Definition Guide* for full details of the REQUESTMODEL resource definition.

Generic pattern matching

OMGINTERFACE, OMGMODULE, and OMGOPERATION can be defined as generic patterns. The rules for pattern matching are summarized as follows:

- Double colons are used as component separators. Each component must be between 1 and 16 characters long.
- Wildcard characters + and * are allowed, matching one (+) or more(*) characters (excluding colons).
- Wildcard '**' matches any number of components of the module name. At most one '**' can be used in a pattern, but it can be used in any position (beginning, middle or end).
- If used, the '*' wildcard character must be the last character of a double-colon-separated component.

If a request is received that matches several generic names, the least generic is selected. The total length of the module pattern may be up to 58 characters. The total length of the interface and operation patterns may be up to 31 characters.

REQUESTMODEL example

This is an example of a generic definition that accepts any OMGMODULE, OMGINTERFACE, and OMGOPERATION. It would act as a default, replacing the supplied default CIOD.

```
DEFINE REQUESTMODEL(GENERIC) GROUP(TEST)
DESCRIPTION(Generic definition for test purposes only)
OMGMODULE(**)
OMGINTERFACE(*)
OMGOPERATION(*)
TRANSID(FRED)
```

Dynamic routing

If the method invocation is to be routed to another region (AOR), you must also define the transaction in the AOR with the PROGRAM parameter set to DFHMIRS and INBFMH set to ALL. You can base this transaction definition on CSMI.

Note that if you are distributing requests running under the CICS supplied transactions CIOD and CIOF, then these transaction definitions must be replaced in the AOR and defined as above. Alternatively, you may specify different default and factory transactions in the TOR, with corresponding definitions in the AOR..

Supplied REQUESTMODEL definitions

The following REQUESTMODEL definition is supplied in group DFH$IIOP, which is included in GRPLIST:

```
DEFINE REQUESTMODEL(DFJ$GFAC)
GROUP(DFH$IIOP)
DESCRIPTION(Generic Factory)
OMGMODULE(org::omg::CosLifeCycle)
OMGINTERFACE(GenericFactory)
OMGOPERATION(*)
TRANSID(CIOF)
```

Obtaining a CICS USERID

You may optionally provide a User Replaceable Module (URM), to examine elements of the incoming IIOP request and generate a USERID to be used when the TRANSID obtained from the selected REQUESTMODEL is started. If you do not specify a URM name in the TCPIPSERVICE, CICS will call DFHXOPUS. If no DFHXOPUS PROGRAM resource definition is installed, or no USERID is supplied, a default CICS USERID is used. You can define this in the CICS system initialization parameter DFLTUSER, or allow it to default to CICSUSER.

DFHXOPUS may use CICS services, such as Task Related User Exits to access DB2, and application parameters encoded within the body of the request. It is run under a generic TRANSID and USERID, whose definitions can be overridden.

A new unit of work is begun using the newly determined USERID and TRANSID to process the client request.

The following COMMAREA is passed to the URM. This structure is based on the format of an IIOP message defined in *The Common Object Request Broker: Architecture and Specification* obtainable from the OMG web site at:

`http://www.omg.org/library`

Offset Hex	Type	Len	Name
(0)	STRUCTURE	60	sXOPUS
(0)	FULLWORD	4	pIIOPData
(4)	FULLWORD	4	lIIOPData
(8)	FULLWORD	4	pRequestBody
(C)	FULLWORD	4	lRequestBody
(10)	FULLWORD	4	pOMGModule
(14	FULLWORD	4	lOMGModule
(18)	FULLWORD	4	pOMGOperation
(1C)	FULLWORD	4	lOMGOperation
(20)	FULLWORD	4	pTRANSID
(24)	FULLWORD	4	lTRANSID
(28)	FULLWORD	4	pUSERID
(2C)	FULLWORD	4	lUSERID
(30)	CHARACTER	1	littleEndian
(31)	CHARACTER	3	reserved
(34)	FULLWORD	4	RETNCODE
(38)	FULLWORD	4	REASCODE

pIIOPData
The address of the unconverted IIOP buffer.

lIIOPData
The length of the unconverted IIOP buffer.

pRequestbody
The address of the incoming IIOP request.

lRequestbody
The length of the incoming IIOP request.

pOMGModule
A pointer to the OMGModule name in EBCDIC.

lOMGModule
The length of the OMGModule name.

pOMGOperation
A pointer to the OMGOperation name in EBCDIC.

lOMGOperation
The length of the OMGOperation.

pTRANSID
A pointer to the TRANSID.

lTRANSID
The length of the TRANSID.

pUSERID
A pointer to the storage area where the USERID is to be returned.

lUSERID
The length of the USERID.

littleEndian
A byte-order indicator, where:

1 indicates little-endian

0 indicates not little-endian

RETNCODE
The return code.

REASCODE
The reason code.

The URM program returns the USERID at the address pUSERID, with lUSERID set to the length (maximum 8 characters). RETNCODE is set to RCUSRID (X'01') if a USERID is being returned. The URM may also change the TRANSID value, but all other fields should be unchanged, or unpredictable results will occur.

See the *CICS Customization Guide* for information about installing user replaceable modules.

Messages greater than 32K

If the request or reply data is less than 32K, it is passed to DFJIIOP in a COMMAREA, but if it is greater than 32K, it is passed in a temporary storage (TS) queue. The queue name used to pass incoming data has the prefix DFIO and the queue name used to return data from the server has the prefix DFJO. You need to define TSMODELs for these prefixes (DFIO and DFJO) to ensure that the TS queues can be accessed from both the TOR and the AOR. These TSMODELs should be defined with LOCATION (AUXILIARY) and RECOVERY(NO) and can reside in a coupling facility or queue owning region (QOR).

If TST=NO is not specified in the system initialization table, the equivalent TST entries should be defined. Note that the TSQPREFIX attribute on the TCPIPSERVICE definition is not used here.

Chapter 32. Developing IIOP applications

Applications are defined as ***interfaces*** and ***operations*** in IDL and implemented in Java. The pieces of an application are:

- The IDL
- A client program that makes calls to the server based on the IDL definition
- A server program that implements the interfaces defined in the IDL
- CICS definitions for the server execution

The CORBA interface and operation names are mapped to corresponding Java implementations. You can develop server implementations that use the CICS Java classes (JCICS) to access CICS services. See the Javadoc HTML documentation for details of the JCICS classes, and the *CICS Application Programming Guide* for an explanation of how to develop server applications using them.

The JAVADOC HTML is stored in the OS/390 UNIX System Services HFS in the directory **$CICS_HOME/docs** during CICS installation. You should transfer this file in binary mode to a workstation, to a file system that can support long names, such as OS/2 HPFS, FAT32 or NTFS, and unzip it. You can then read it using a web browser. The following file is supplied:

dfjcics_docs.zip

This section tells you how to prepare the parts of the application. It covers the following topics:

The Interface Definition Language (IDL)

Before you write a CORBA client or server application, you must first create an OMG IDL file that contains the definitions of interfaces the server implementation will support. An OMG IDL file describes the data-types, operations, and objects that the client can use to make a request, and that a server must provide for an implementation of a given object.

For information about writing IDL, see the OMG publication, *Common Object Broker: Architecture and Specification*, obtainable from the OMG web site at:

`http://www.omg.org/library`

You process the IDL definitions with an IDL to Java compiler (sometimes called a parser or generator). You must use a compiler provided by the server environment to generate server-side skeletons and helper classes, and a compiler provided by the client environment to generate client-side stub (sometimes called proxy) and helper classes.

The proxies and skeletons provide the object-specific information needed for an ORB to distribute a method invocation.

Note: The CICS compiler provided in **dfjcidl.jar** *must* be used to generate the server-side skeleton and helper classes, otherwise errors will occur during build or execution. If you are running both client and server IDL compilers on the same workstation, you must ensure that CLASSPATH will locate the correct compiler in each case, and that the output is written to separate directories.

The following diagram shows how the same IDL file is used to generate different classes used by the client and the server.

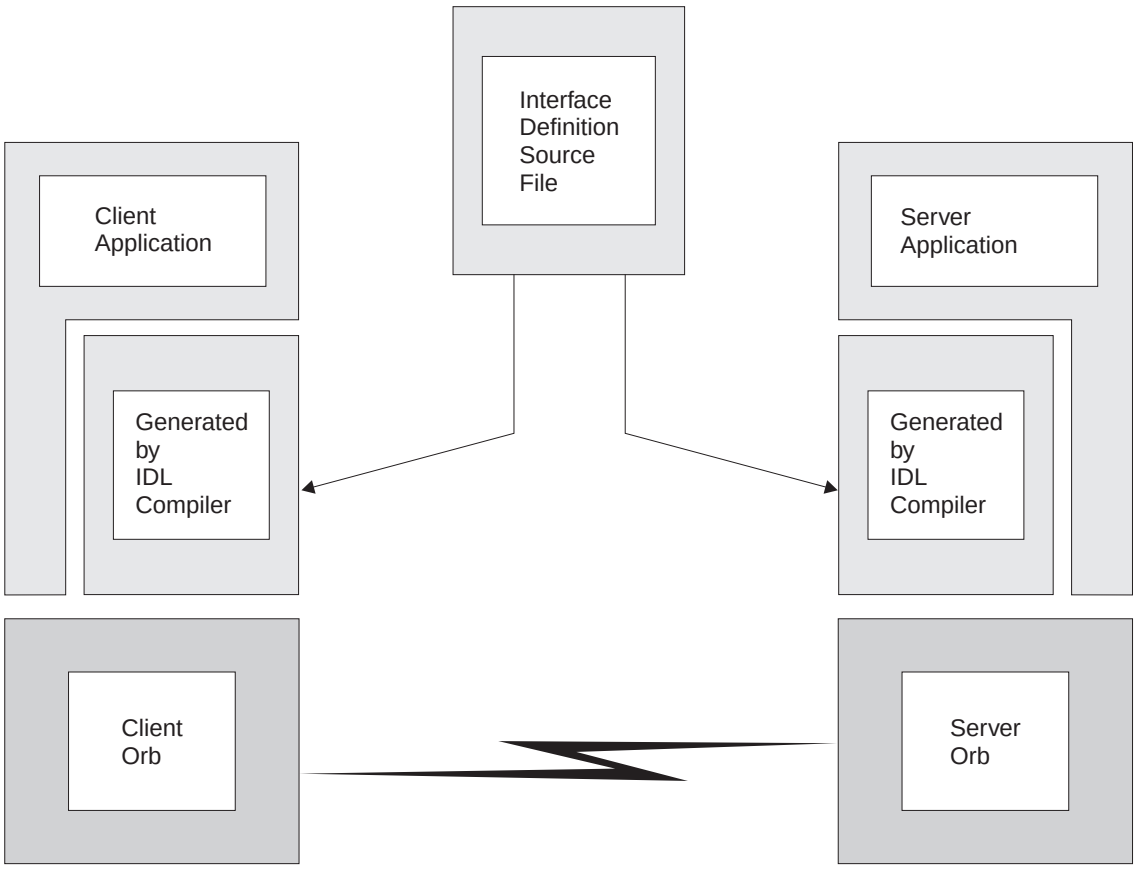


Figure 25. IDL and generated code

Programming model

From the client point of view, an object in a CICS IIOP ORB is just a collection of methods, that is, a stateless object. Each method will represent a piece of logic that may make one or more CICS API calls, including CICS LINKs, to existing CICS programs. At the end of the method, no data is stored in attributes.

This implies that every method must be passed sufficient information in its parameter list to enable it to complete its work. No information is passed to the server by virtue of the object reference, except the object type, which is used to find the class and methods of the implementation. The methods of the object may save

state in an application managed datastore between invocations. They will need to ensure that sufficient information is passed as parameters to subsequent methods so that the saved state can be retrieved.

In order to access a server object, a reference to it is required. In a distributed environment, an object reference, known as an Interoperable Object Reference (IOR), is more than just a storage address obtained using `new`. It contains enough information to allow:

- a request to be directed to the correct server (host, port number)
- an object to be located or created (classname, instance data)

IORs may be returned by server methods, but a factory class is needed to create an initial IOR.

An implementation of the CosLifeCycle GenericFactory is provided for object creation. (Note that for a stateless object, the GenericFactory is completely adequate; there is no value in allowing more powerful factories such as application specific factories). A utility, **GenFacIOR** is provided to create a stringified IOR of the GenericFactory class.

Due to the stateless nature of the object, there is seldom any point in a client creating more than one instance of a class. Once a client has created an instance of an object, for example `bankaccountfacilitator`, the same object can be used to access both Mr X's account and Mr Y's account; the account number is an input parameter in every method.

Note: We have called the object in this example a `bankaccountfacilitator` so that it can perform actions on any account. To have called it simply a `bankaccount` might imply that the instance always represented Mr X's account.

In the server programming model, each method is a subroutine. The parameters passed allow you to establish temporary variables from various existing databases or applications, to perform business logic, to store data in the existing databases or applications, and to return results when the subroutine returns.

Developing the server program

The server program can be developed on any platform that supports Java. For example, an NT workstation, AIX or the OS/390 UNIX System Services environment of ESA. The following steps are required:

1. Write the IDL definition of the interfaces and operations that form your application.
2. Compile the IDL file to generate CORBA skeleton and helper classes using the compiler provided by CICS.

   The IDL compiler can be invoked (with dfjcidl.jar in your CLASSPATH) as follows:

   ```
   java com.ibm.idl.toJava.Compile [options] <idl file>
   ```

   Where <idl file> is the name of the file containing the IDL definitions, and [options] is any combination of the following options, which may appear in any order. <idl file> is required and must appear last. At least **-f** must be specified.

   -d<symbol>
   The equivalent of the following line in an IDL file: `#define <symbol>`

-emitAll
Emit all types, including those found in `#included` files.

-f<side>
Define the bindings to emit. <side> can be:

client not applicable to CICS.

server does not generate sufficient classes for normal use.

all emits all bindings.

serverTIE
not supported in CICS.

allTIE not supported in CICS

If this option is not specified, then **-fclient** is assumed. In most cases you should use **-fall**.

-i<include path>
Add another directory. By default, the current directory is scanned for included files.

-keep If a file to be generated already exists, do not overwrite it. By default it is overwritten.

-m Generate information to be included in a **make** description file; output goes to a **.u** file.

-sep <string>
Replace the file separator character with <string> in the file names listed in the **.u** file , if **-m** is specified.

-pkgPrefix <t> <pkg>
Make sure that wherever the type or module <t> is encountered, it resides within <pkg> in all generated files. <t> is a fully qualified Java-style name.

-v Verbose mode.

-bean Generate classes that can be used as Java beans.

-stateful
Parse stateful interface objects (used for Objects-by-value). Note that this is non-standard IDL and is not supported by CICS.

3. Write your server implementation in Java. The idl compiler will generate an abstract class called_*interfacename***ImplBase**. Your program must extend this. If objects of this type are to be created by the Generic Factory, it must be called _*interface name*Impl. For example:

   ```
   public class _BankAccountImpl extends _BankAccountImplBase
   ```

   This requires the CORBA classes from **dfjcorb.jar** and may use the CICS API Java classes from **dfjcics.jar** supplied by CICS. See the *CICS Application Programming Guide* for information about CICS support for Java programs.
4. Compile your program and the output from step 2, with the javac compiler or an equivalent, such as VisualAge for Java, with the following files in your CLASSPATH:
   - **dfjcorb.jar**
   - **dfjcics.jar** (if required)
5. If you are developing your program on a workstation, transmit the output from step 4 to the OS/390 UNIX System Services (OpenEdition) environment in ESA.

6. The Java bytecode can then be processed by the binder provided by the VisualAge for Java, Enterprise ToolKit for OS/390 to produce a Java program object that can be loaded and run by CICS. See the *CICS Application Programming Guide* for information about using the VisualAge for Java, Enterprise ToolKit for OS/390.

IDL example

The following example describes a bank account whose contents can be queried and updated. Note that this example has a parameter that identifies the instance of the BankAccount, to satisfy the 'stateless' restriction. The following IDL defines the interface and operations:

```
module bank {
 struct BankData {
  long acnum;
  string custname;
  string custaddr;
  long balance;
};

// this interface is used to manage the bank accounts
interface BankAccount {
  exception ACCOUNT_ERROR { long errcode; string message;};

  // query methods
  long querybalance(in long acnum) raises (ACCOUNT_ERROR);
  string queryname(in long acnum) raises (ACCOUNT_ERROR);
  string queryaddress(in long acnum) raises (ACCOUNT_ERROR);

  // setter methods
  void setbalance(in long acnum, in long balance) raises (ACCOUNT_ERROR);
  void setaddress(in long acnum, in string address) raises (ACCOUNT_ERROR);
};
};
```

In this example, the module name is `bank`, the interface name is `BankAccount` and the Operations are `querybalance`, and `setbalance`.

Server implementation

The server implementation of the above IDL must be called `_BankAccountImpl` if objects of this type are to be created by the GenericFactory and must extend `_BankAccountImplBase`, which is generated by the IDL compiler. It is part of the Java package `bank`. You can see full details of this implementation in the BAnkAccount sample application distributed in $CICS_HOME/samples/dfjcorb

Resource definition for example

The following REQUESTMODEL example associates the inbound request with a TRANSID that gives the request the right execution characteristics.

```
DEFINE REQUESTMODEL(DFJIIBS)
       GROUP(DFH£IIOP)
       DESCRIPTION(Bank account sample)
       OMGMODULE(bank)
       OMGINTERFACE(BankAccount*)
       OMGOPERATION(*)
       TRANSID(BNKS)
```

The BNKS transaction defines execution characteristics for query and update requests received using IIOP. It runs the DFHIIOPA program that links to DFJIIOP, which invokes the methods in `_BankAcctImpl`.

Developing the client program

1. Process the IDL file with an IDL to Java compiler suitable for your client system (using the same IDL file that you used to build the server application).
2. Create a stringified object reference to the GenericFactory using the GenFacIOR utility described in “The GenFacIOR utility”.
3. Write your client program, containing calls to the server. To obtain an initial object reference, use the GenericFactory as shown in “Client example”.
4. Compile the client program, and the output from step1, with javac or an equivalent compiler.

Note: You need **dfjcorb.jar** in the CLASSPATH when generating server side (CICS) applications, and your client ORB vendor’s classes in the CLASSPATH when generating client side applications.

The GenFacIOR utility

The GenFacIOR utility is a Java class used to generate a stringified IOR for a GenericFactory on a given host and port. It stores the generated IOR in a file named **genfac.ior**. GenFacIOR generates a reference to `org.omg.CosLifeCycle._GenericFactoryImpl`. To create the IOR you must:

1. Ensure that **dfjcorb.jar** is in your classpath.
2. Use:

```
 java com.ibm.cics.server.ts.iiop.GenFacIOR -d  <directory>
-host <hostname> -port <port>
```

   Where:

   directory
   is the destination for the IOR file. This defaults to the current directory.

   hostname
   is the string name used to identify the host. For example, winmvs2c.hursley.ibm.com.

   port is the port number of the TCPIPSERVICE. It defaults to 535.
3. Use this file in the client to get the IOR. If you generate this file in OS/390 UNIX System Services, then you must transfer it to the client, or make it available as a **text** file.

If you need a client to access more than one TCP/IP port or use more than one CICS region, you will need to generate an IOR for each host/port combination you are intending to use. To keep the IORs separate, you will either need to rename the generated file or place them in different directories.

Client example

The following example shows how the GenericFactory service is used by a client program to create an **account** object. The client must first create a proxy for the GenericFactory.

Java bindings for part of the CORBA CosLifeCycle and CosNaming modules are required. If they are not provided by the client ORB, then you can build them using the client ORB's IDL to Java compiler, from the IDL given in the CORBA specification, or alternatively, use the IDL subset provided in $CICS_HOME/samples/dfjcorb. The following example, and the supplied samples, require bindings that can be imported as `org.omg.CosNaming` and `org.omg.CosLifeCycle`.

In order to create an account object, the client must first create a proxy for the GenericFactory. The following example assumes that a stringified reference to the GenericFactory exists in a file available to a client, and is returned by the **getFactoryIOR()** method.

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosLifeCycle.*;
import org.omg.CosNaming.*;
public class bankLineModeClient{

//The following method reads the ior from a file and returns it in the string
 String factoryIOR = getFactoryIOR();
// Turn the stringified reference into the proxy
 org.omg.CORBA.Object genFacRef = orb.string_to_object(factoryIOR);
// narrow to correct interface
 GenericFactory fact = GenericFactoryHelper.narrow(genFacRef);
```

Now that the client has a generic factory, it can use it to create an **account** object.

```
// The Generic factory needs a key, which is a sequence of namecomponents
 NameComponent nc = new NameComponent("bank::BankAccount","object interface");
 NameComponent key[] = {nc};
//The  Generic factory also requires criteria (which it ignores)
 NVP mycriteria[] = {};

Now create the object
 org.omg.CORBA.Object objRef = fact.create_object(key, mycriteria);
// and narrow to correct interface
BankAccount acctRef = BankAccountHelper.narrow(objRef);
```

Now the client has an object, it can use it:

```
int ac1 = 1234; // Tony's account
int ac2 = 3456; // Lou's account
String name;
String address;
int balance;


try {
  name=acctRef.queryname(ac1);
  System.out.println("a/c num:"+ac1+" name:"+name");
}
catch (exception e) {
  System.err.println("query error");
}
```

Note: NVP (Name Value Pair) is a datatype defined in the CORBA IDL for the Generic Factory interface.

Chapter 33. IIOP sample applications

Two sample application that use IIOP and the CICS Java programming support are shipped with CICS. These sample programs are designed to run using the VisualAge for Java, Enterprise ToolKit for OS/390 to bind the server programs into Java program objects that can be loaded and run by CICS.

The following sample applications are provided:

HelloWorld sample

This sample provides a simple test of the IIOP components. The client program:

- reads the file genfac.ior to obtain a reference to the generic factory
- uses the generic factory to create a HelloWorld object
- invokes method `sayHello` to send a greeting to the server (Hello from HelloWorldClient)and receive a greeting from it in reply (Hello from CICS TS)

The design of the application is described in comments in the code.

BankAccount sample

The sample consists of the following main parts:

1. A traditional CICS application that uses BMS and the EXEC CICS API, written in C. This application consists of two transactions:

   BNKI Initializes a file with information about a number of bank accounts. These accounts have numbers in the range 23 through 30.

   BNKQ Queries the information in the accounts. There is also a CICS program, DFH$IICC, which performs a credit check for an account.
2. An implementation of an IDL interface that defines a bank account object. The implementation is written in Java and runs as a CORBA server object inside CICS. This implementation uses the bank account file to access bank account information and the DFH$IICC credit check program to obtain credit ratings.
3. A CORBA client application written in Java that displays information about bank account objects.

The design of the application is described in comments in the code.

This chapter describes the samples and tells you how to run them. The following topics are covered:

Requirements to run the samples

This section describes the specific requirements to run the sample applications, in addition to those described in “Chapter 30. Requirements for IIOP applications” on page 173.

The sample Java source and makefiles are stored in the OS/390 UNIX System Services HFS during CICS installation, in the following directories:

- $CICS_HOME/samples/dfjcorb/HelloWorld
- $CICS_HOME/samples/dfjcorb/BankAccount

$CICS_HOME is an environment variable defining the installation directory prefix:

```
/usr/lpp/cicsts/<username>
```

Where **username** is a name you can choose during the installation of CICS, defaulting to cicsts13.

The following CICS C language programs used by the BankAccount sample are stored in SDFHSAMP during CICS installation.

DFH$IIBI
C program that initializes the BANKACCT file. Run by the BNKI transaction.

DFH$IIBQ
C program that queries the accounts held in BANKACCT.

DFH$IICC
C program that performs a credit check. This is called by DFH$IIBQ.

DFH$IIMA
BMS mapset BANKINQ.

DFH$IIQR
Bank Query structure

DFH$IICH
Credit Check Structure

DFH$IIAT
Acctrec structure

Note: In the names of sample programs and files described in this book, the dollar symbol ($) is used as a national currency symbol and is assumed to be assigned the EBCDIC code point X'5B'. In some countries a different currency symbol, for example the pound symbol (£), or the yen symbol (¥), is assigned the same EBCDIC code point. In these countries, the appropriate currency symbol should be used instead of the dollar symbol.

Resource definitions

CICS resource definitions for the sample applications are supplied in group DFH$IIOP. This contains resource definitions required for the HelloWorld sample:

- DFJ$IIHE PROGRAM definition
- IIHE TRANSACTION definition
- DFJIIHE REQUESTMODEL definition

and resource definitions required for the BankAccount sample:

- DFH$IIBI PROGRAM definition
- DFH$IIBQ PROGRAM definition
- DFJ$IIBS PROGRAM definition
- DFH$IICC PROGRAM definition
- BANKINQ MAPSET definition
- BNKI TRANSACTION definition
- BNKQ TRANSACTION definition
- BNKS TRANSACTION definition
- BANKACCT FILE definition
- DFJIIBS REQUESTMODEL definition

Installing CICS resources

The CICS supplied group DFH$IIOP must be installed before you run the sample. Do this by including the group DFH$IIOP in GRPLIST before starting CICS or by using the CEDA option INSTALL to install the resources in CICS whilst it is running. See the *CICS Supplied Transactions* for information about using CEDA to install resource definitions.

Generic Factory

Java bindings for part of the CORBA CosLifeCycle and CosNaming modules are required. If they are not provided by the client ORB, then you can build them using the client ORB's IDL to Java compiler, from the IDL given in the CORBA specification, or alternatively, use the IDL subsets provided in $CICS_HOME/samples/dfjcorb/.

Note: You may need to change the **import** statements in the client code to correspond with the package name of the bindings generated by your ORB's IDL compiler. Alternatively, use your client ORB IDL compiler's equivalent of the **-pkgPrefix** option to set the package name to that required by the Java program's import statement.

You will need to create a **genfac.ior** file containing an object reference to your server's generic factory, and place it in the current directory.

CICS libraries

You will need to add $LIB_PREFIX.LOAD to the DFHRPL concatenation of your CICS start-up jobstream. (Where $LIB_PREFIX is your PDSE dataset name prefix).

The HelloWorld sample

This section tells you what you need to do to run the HelloWorld sample application. It covers the following topics:

Building the server side HelloWorld application

The makefile in $CICS_HOME/samples/dfjcorb/HelloWorld/server builds everything required for the server side application. Before you can build the sample, you need to:

1. Set up the following environment variables:

   $LIB_PREFIX
   Your PDSE dataset name prefix

   $CICS_HOME
   The installation directory prefix of CICS TS.

   $JAVA_HOME
   The installation directory prefix of the JDK.
2. Allocate a PDSE called $LIB_PREFIX.LOAD.

To build the programs, enter the following command from $CICS_HOME/samples/dfjcorb/HelloWorld/server:

```
make
```

This makes DFJ$IIHE, the Java server program that implements the HelloWorld object.

Building the client side HelloWorld application

$CICS_HOME/samples/dfjcorb/HelloWorld/client contains the CORBA client part of the application. The source of the Java client application is called **HelloWorldClient.java**. This application should run with any CORBA-compliant ORB.

The following steps are required to build the Java client application:

1. Download the following files to the client workstation:
   - .../dfjcorb/HelloWorld/HelloWorld.idl
   - .../dfjcorb/HelloWorld/client/HelloWorldClient.java
2. Compile the provided IDL with the client ORB's IDL-to-Java compiler to produce the Java client side stubs required by the sample application.
3. Compile the client application, ensuring that the Java classes produced in the previous step are available through the CLASSPATH environment variable.

Running the HelloWorld sample application

Run the client application using:

```
java HelloWorldClient
```

The BankAccount sample

This section tells you what you need to do to run the BankAccount sample application. It covers the following topics:

- "Building the server side BankAccount application" on page 193
- "Building the client side BankAccount application" on page 193
- "Running the BankAccount sample application" on page 194

Create the VSAM file

Define the VSAM file to hold the bank account data, using the following IDCAMS parameters:

```
DEFINE    CLUSTER (                          -
                NAME (CICS530.BANKACCT )     -
                CYLINDERS(01)                -
                REUSE                        -
                KEYS(4 0)                    -
                RECORDSIZE(168 168))
```

Prepare CICS programs

Translate, compile and link the CICS sample programs:

- DFH$IIBI
- DFH$IIBQ
- DFH$IICC

Prepare BMS maps

The file DFH$IIMA contains one mapset BANKINQ with two maps. Compile and link the mapset BANKINQ.

Building the server side BankAccount application

The makefile in $CICS_HOME/samples/dfjcorb/BankAccount/server builds everything required for the CORBA part of the server side application. Before you can build the sample, you need to:

1. Set up the following environment variables:

   $LIB_PREFIX
   Your PDSE dataset name prefix

   $CICS_HOME
   The installation directory prefix of CICS TS.

   $JAVA_HOME
   The installation directory prefix of the JDK.
2. Allocate a PDSE called $LIB_PREFIX.LOAD (or change the name by editing the LM macro in $CICS_HOME/samples/dfjcorb/BankAccount/server/Makefile.bank)

To build the programs, enter the following command from $CICS_HOME/samples/dfjcorb/BankAccount/server:

```
make
```

This makes DFH$IIBS, the Java server program that implements the bank account object.

Building the client side BankAccount application

$CICS_HOME/samples/dfjcorb/BankAccount/javaclient contains the CORBA client part of the application. The source of the Java client application is called **bankLineModeClient.java**. This application should run with any CORBA-compliant ORB.

The following steps are required to build the Java client application:

1. Download the following files to the client workstation:
   - .../dfjcorb/BankAccount/BankAccount.idl
   - .../dfjcorb/BankAccount/javaclient/bankLineModeClient.java
2. Compile the provided IDL with the client ORB's IDL-to-Java compiler to produce the Java client side stubs required by the sample application.
3. Compile the client application, ensuring that the Java classes produced in the previous step are available through the CLASSPATH environment variable.

Running the BankAccount sample application

The following steps are required to run the sample application:

1. Run the BNKI CICS transaction to load data into the account file.
2. Run the client application using:

```
java bankLineModeClient
```

Part 7. 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	DFHWBBLL

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 5. 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** **ssl_keysize** **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_eyecatcher** to indicate which mode of operation is to be used.

For information about writing a converter for the business logic interface, see "Chapter 8. Writing a converter" on page 49.

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 211, but as described in "Responses" on page 202 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 in 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 in not necessarily the same as `wbbl_indata_ptr`.

wbbl_prolog_size
(Input only)

A halfword binary number that must be set to 56 (that is, the length of the `wbbl_prolog` substructure).

wbbl_resource_length
(Input only)

A fullword binary number that must contain the length of the URI resource that is being requested (that is, the non-network part of the URL, starting at the first 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_ssl_keysize
(Input only)

The size of the encryption key negotiated during the SSL handshake, if secure sockets layer is being used. It contains zero if SSL is not being used.

wbbl_status_size
(Input only)

A 1–byte binary field that must be set to the length of the `wbbl_status` substructure.

wbbl_user_data_length
(Input only)

A fullword binary number that must be set to the length of the 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 6. 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_length**	**user_data_ptr** **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 36. 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 208.

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 208.

wbra_server_ip_address
(Input only)

The 32-bit IP address of the OS/390 eNetwork Communications Server region 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 **wbbl_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
The server controller 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 OS/390 eNetwork Communications Server socket is closed.

URP_INVALID
The alias transaction is not started. The server controller writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, response 400 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the OS/390 eNetwork Communications Server 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 OS/390 eNetwork Communications Server socket is closed.

If you return any other value in **wbra_response**, the server controller writes an exception trace entry (trace point 4510), and issues a message (DFHWB0523). If the request is an HTTP request, response 400 is sent to the web browser. If the request is not an HTTP request, no response is sent, and the OS/390 eNetwork Communications Server 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 90 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 “/”.

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

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

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

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

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** function of the converter
- Reference information for the **Encode** function 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 7. Parameters for **Decode**

Input decode_	Inout decode_	Output decode_
client_address **client_address_string** **eyecatcher** **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** **server_program** **user_token**	**input_data_len** **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).

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_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
(Output only)

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 32K.

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 32K.

decode_reason
(Output only)

A reason code—see “Responses and reason codes” on page 215.

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 215.

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 can be either binary zero (X’00’), indicating a pre-CICS TS 1.3 version parameter list, or a character zero (X’F0’), indicating a CICS TS 1.3 version parameter list.

decode_volatile
(Input)

A single-character code indicating whether the data area pointed to be **decode_data_ptr** can be 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_pt**r 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 OS/390 eNetwork Communications Server 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 MVS 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 OS/390 eNetwork Communications Server 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 OS/390 eNetwork Communications Server 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 OS/390 eNetwork Communications Server 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 OS/390 eNetwork Communications Server 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 90 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 8. Parameters for **Encode**

Input encode_	Inout encode_	Output encode_
eyecatcher **function** **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_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" on page 218.

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 can be either binary zero (X’00’), indicating a pre-CICS TS 1.3 version parameter list, or a character zero (X’F0’), indicating a CICS TS 1.3 version parameter list.

encode_volatile
(Input)

A single-character code indicating whether the data area pointed to be **encode_data_ptr** can be 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_pt**r 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 OS/390 eNetwork Communications Server 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.

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 OS/390 eNetwork Communications Server 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 90 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 an MVS partitioned data set (PDS), or can be provided inline in your application program, or can be defined in a DOCTEMPLATE definition. DOCTEMPLATEs definte 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 definition for the PDS member in the DFHHTML DDname 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 9. 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 90 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 224.

wbtl_reason
(Output only)

Might contain additional information about an error for some responses. See “Responses and reason codes” on page 224.

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 the PDS, 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 PDS member specified by wbtl_tempate_name. This is the date and time (in CICS ABSTIME format) when the template was last modified, if the modification was made with the ISPF editor. Otherwise it 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 the PDS, 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 the PDS, 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 "Symbols, symbol table, and symbol list" on page 76. 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 PDS.

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 90 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_**. A complete list of HTTP headers can be found at http://ds.internic.net/rfc/rfc1945.txt. 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 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 “Sample programs for security” on page 84.

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 10. 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 10. 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 that 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 DFHWBEPC being passed by CICS.

Appendix I. Programming reference information for the CICS Transaction Gateway for OS/390

This chapter contains General-use Programming Interface and Associated Guidance Information.

The reference sections are as follows:

- "The JavaGateway class"
- "The ECIRequest class" on page 242
- "The Callbackable class" on page 251
- "The GatewayRequest class" on page 252
- "The CicsCpRequest class" on page 252

The information for each class is organized as follows:

- Inheritance diagram, and a general description of the class
- Public variables of the objects of the class
- Constants of the class
- Constructors for objects
- Methods for the class. The information for each method is organized as follows:
  - A Java declaration is accompanied by a general description of the method.
  - **Parameters** lists the parameters you must supply for the method.
  - **Returns** describes the object returned by the method.
  - **Throws** lists the exceptions that can be raised by the method.
  - A programming fragment shows how the method can be used.

The descriptions of the variables, constants, constructors, and methods for the `ECIRequest` and `EPIRequest` classes rely heavily on a knowledge of the CICS external call interface (ECI) and external presentation interface (EPI). These interfaces are described in *CICS Family: Client/Server Programming*, and you should have a copy of that manual available for reference. The typographic conventions of that manual are used in this chapter when elements of the ECI and EPI are being described.

Some of the functions and facilities described here are not supported by the CICS Transaction Gateway for OS/390. See "Making EPI calls" on page 153 and "Making ECI calls" on page 151 for details.

The JavaGateway class

```
java.lang.Object
   |
   +----ibm.cics.jgate.client.JavaGateway
```

public class `JavaGateway`

extends `Object`

implements `Runnable`

This class manages the connection between a Java program and a CICS Transaction Gateway or a local CICS Client. You need a JavaGateway object for each CICS Transaction Gateway or local CICS Client that you wish to communicate with. Once a connection has been established, you use the flow method to send GatewayRequest objects, or subclasses of GatewayRequest, to the corresponding CICS Transaction Gateway or local CICS Client.

There are three kinds of JavaGateway object:

network
A network JavaGateway object manages a connection between a program and a remote CICS Transaction Gateway. The connection between the computer running the Java application or applet and the remote Gateway is through TCP/IP, and so when creating a network JavaGateway object you must supply the TCP/IP address and port of the remote CICS Transaction Gateway.

local A local JavaGateway object communicates directly to a locally installed CICS Client. No network connection is used between the Java program and the CICS Client, and no CICS Transaction Gateway required. Given the requirement to communicate to a locally installed CICS Client, a local JavaGateway object is generally only applicable for use in a Java application or servlet.

automatic
An automatic JavaGateway object determines at run-time whether to act as a network or local JavaGateway. If you wish to create an automatic JavaGateway object, a TCP/IP address must be specified. If at run-time this TCP/IP address matches the local host address of the computer running the program, then the JavaGateway object behaves as if it were a local JavaGateway object, otherwise it behaves as a network JavaGateway object.

A JavaGateway object has a second thread which is used to listen for replies from the CICS Transaction Gateway or the local CICS Client.

Constructors

There is only one constructor for a JavaGateway object:

```
public JavaGateway(String strServer,
                   int iPort) throws IOException
```

This constructor creates a JavaGateway object, and connects it to the CICS Transaction Gateway at the specified IP address and port. Creating a new JavaGateway object causes the creation of a background listener thread associated with that JavaGateway object.

Parameters

strServer
A string specifying the Gateway address. This string can have one of four forms:

- tcp://*addr*/ creates a network JavaGateway object connected to the CICS Transaction Gateway at the TCP/IP address *addr*.
- local: creates a local JavaGateway object.
- auto://*addr* creates an automatic JavaGateway object that uses the TCP/IP address *addr* to determine whether it is a network JavaGateway object or a local JavaGateway object.

- *addr* creates a network JavaGateway object connected to the CICS Transaction Gateway at the TCP/IP address *addr*.

iPort A number specifying the Gateway port

The following code fragment shows the creation of a JavaGateway object jg with literals for the parameters.

```
JavaGateway jg = new JavaGateway("gateway8", 2006);
```

Methods

The following methods of a JavaGateway object are described:

- flow
- close
- LocalJavaGateway.destroy

flow

```
public int flow(GatewayRequest gatRequest) throws IOException
```

Flows the specified GatewayRequest object to the CICS Transaction Gateway and then waits for the reply. The reply is returned in the original request object.

Parameters

gatRequest
The GatewayRequest object containing the request

Returns
Return code from this flow operation

Throws
IOException if a network I/O error occurs during the operation

The following code fragment shows how you can use the flow method of the JavaGateway object jg with a previously defined ECIRequest object eciRequest.

```
jg.flow(eciRequest);
```

close

```
public synchronized void close() throws IOException
```

Closes the connection to the CICS Transaction Gateway. If there were any other threads using this JavaGateway object that were in the middle of a flow operation, they will return with an error.

Throws
IOException if a network I/O error occurs during the operation

The following code fragment shows how you can close the JavaGateway object jg.

```
jg.close();
```

LocalJavaGateway.destroy

```
public static void LocalJavaGateway.destroy() throws IOException
```

Prevents the creation of new local JavaGateway objects.

Throws
IOException if a network I/O error occurs during the operation

The ECIRequest class

```
java.lang.Object
   |
   +----ibm.cics.jgate.client.GatewayRequest
           |
           +----ibm.cics.jgate.client.ECIRequest
```

public class `ECIRequest`

extends `GatewayRequest`

This class contains the details of the ECI request to the CICS Transaction Gateway. To execute the request, you use the `flow` method of a previously defined `JavaGateway` object to pass the `ECIRequest` object to the CICS Transaction Gateway, or to a local CICS client.

Public variables

Table 11 shows the public variables for an `ECIRequest` object. Other variables are accessible only by the methods. Each variable is related to a field or a parameter in the ECI interface described in *CICS Family: Client/Server Programming*. Figures in parentheses refer to the notes after the table.

Table 11. Public variables in an ECIRequest object

Java declaration	Meaning
`public String Abend_Code`	**eci_abend_code** (ECI parameter block) (2)
`public int Call_Type`	**eci_call_type** (ECI parameter block)
`public int Cics_Rc`	**eci_return_code** (ECI parameter block)
`public int CicsClientStatus`	**CicsClientStatus** (ECI status block) (1)
`public int CicsServerStatus`	**CicsServerStatus** (ECI status block) (1)
`public byte Commarea[]`	**eci_commarea** (ECI parameter block)
`public int Commarea_Length`	**eci_commarea_length** (ECI parameter block)
`public int ConnectionType`	**ConnectionType** (ECI status block) (1)
`public int Extend_Mode`	**eci_extend_mode** (ECI parameter block)
`public int Luw_Token`	**eci_luw_token** (ECI parameter block)
`public int Message_Qualifier`	**eci_message_qualifier** (ECI parameter block)
`public String Password`	**eci_password** (ECI parameter block)
`public String Program`	**eci_program_name** (ECI parameter block)
`public String Server`	**eci_system_name** (ECI parameter block)
`public Vector SystemList`	**List** parameter of **CICS_EciListSystems** call
`public String Transid`	**eci_transid** (ECI parameter block)
`public String Userid`	**eci_userid** (ECI parameter block)

Notes:

1. Flowing an `ECIRequest` object to a CICS Transaction Gateway will set these variables only if the call type is `ECI_STATE_SYNC_JAVA` or `ECI_STATE_ASYNC_JAVA`. You should use these call types for status request calls rather than `ECI_STATE_SYNC` and `ECI_STATE_ASYNC`.
2. This field is used when you use the CICS Transaction Gateway to communicate with CICS TS and the program being called abends.

Constants

Table 12 shows the constants of the `ECIRequest` class. Most of the constants are related to a constant of the ECI interface described in *CICS Family: Client/Server Programming*. Figures in parentheses refer to the notes after the table.

Table 12. Constants of the ECIRequest class

Java declaration	Meaning
`public final static int ECI_ASYNC`	ECI_ASYNC (5)
`public final static int ECI_ASYNC_TPN`	ECI_ASYNC (6)
`public final static int ECI_BACKOUT`	ECI_BACKOUT
`public final static int ECI_CANCEL`	ECI_CANCEL
`public final static int ECI_CLIENTSTATE_INAPPLICABLE`	ECI_CLIENTSTATE_INAPPLICABLE
`public final static int ECI_CLIENTSTATE_UNKNOWN`	ECI_CLIENTSTATE_UNKNOWN
`public final static int ECI_CLIENTSTATE_UP`	ECI_CLIENTSTATE_UP
`public final static int ECI_COMMIT`	ECI_COMMIT
`public final static int ECI_CONNECTED_NOWHERE`	ECI_CONNECTED_NOWHERE
`public final static int ECI_CONNECTED_TO_CLIENT`	ECI_CONNECTED_TO_CLIENT
`public final static int ECI_CONNECTED_TO_SERVER`	ECI_CONNECTED_TO_SERVER
`public final static int ECI_ERR_ALREADY_ACTIVE`	ECI_ERR_ALREADY_ACTIVE
`public final static int ECI_ERR_CALL_FROM_CALLBACK`	ECI_ERR_CALL_FROM_CALLBACK
`public final static int ECI_ERR_CICS_DIED`	ECI_ERR_CICS_DIED
`public final static int ECI_ERR_INVALID_CALL_TYPE`	ECI_ERR_INVALID_CALL_TYPE
`public final static int ECI_ERR_INVALID_DATA_AREA`	ECI_ERR_INVALID_DATA_AREA
`public final static int ECI_ERR_INVALID_DATA_LENGTH`	ECI_ERR_INVALID_DATA_LENGTH
`public final static int ECI_ERR_INVALID_EXTEND_MODE`	ECI_ERR_INVALID_EXTEND_MODE
`public final static int ECI_ERR_INVALID_TRANSID`	ECI_ERR_INVALID_TRANSID
`public final static int ECI_ERR_INVALID_VERSION`	ECI_ERR_INVALID_VERSION
`public final static int ECI_ERR_LUW_TOKEN`	ECI_ERR_LUW_TOKEN

Table 12. Constants of the ECIRequest class (continued)

Java declaration	Meaning
`public final static int ECI_ERR_MAX_SESSIONS`	ECI_ERR_MAX_SESSIONS
`public final static int ECI_ERR_MAX_SYSTEMS`	ECI_ERR_MAX_SYSTEMS
`public final static int ECI_ERR_MORE_SYSTEMS`	ECI_ERR_MORE_SYSTEMS
`public final static int ECI_ERR_NO_CICS`	ECI_ERR_NO_CICS
`public final static int ECI_ERR_NO_REPLY`	ECI_ERR_NO_REPLY
`public final static int ECI_ERR_NO_SESSIONS`	ECI_ERR_NO_SESSIONS
`public final static int ECI_ERR_NO_SYSTEMS`	ECI_ERR_NO_SYSTEMS
`public final static int ECI_ERR_NULL_MESSAGE_ID`	ECI_ERR_NULL_MESSAGE_ID
`public final static int ECI_ERR_NULL_SEM_HANDLE`	ECI_ERR_NULL_SEM_HANDLE
`public final static int ECI_ERR_NULL_WIN_HANDLE`	ECI_ERR_NULL_WIN_HANDLE
`public final static int ECI_ERR_REQUEST_TIMEOUT`	ECI_ERR_REQUEST_TIMEOUT
`public final static int ECI_ERR_RESOURCE_SHORTAGE`	ECI_ERR_RESOURCE_SHORTAGE
`public final static int ECI_ERR_RESPONSE_TIMEOUT`	ECI_ERR_RESPONSE_TIMEOUT
`public final static int ECI_ERR_ROLLEDBACK`	ECI_ERR_ROLLEDBACK
`public final static int ECI_ERR_SECURITY_ERROR`	ECI_ERR_SECURITY_ERROR
`public final static int ECI_ERR_SYSTEM_ERROR`	ECI_ERR_SYSTEM_ERROR
`public final static int ECI_ERR_THREAD_CREATE_ERROR`	ECI_ERR_THREAD_CREATE_ERROR
`public final static int ECI_ERR_TRANSACTION_ABEND`	ECI_ERR_TRANSACTION_ABEND
`public final static int ECI_ERR_UNKNOWN_SERVER`	ECI_ERR_UNKNOWN_SERVER
`public final static int ECI_EXTENDED`	ECI_EXTENDED (1)
`public final static int ECI_GET_REPLY`	ECI_GET_REPLY
`public final static int ECI_GET_REPLY_WAIT`	ECI_GET_REPLY_WAIT
`public final static int ECI_GET_SPECIFIC_REPLY`	ECI_GET_SPECIFIC_REPLY
`public final static int ECI_GET_SPECIFIC_REPLY_WAIT`	ECI_GET_SPECIFIC_REPLY_WAIT
`public final static int ECI_LUW_NEW`	zero
`public final static int ECI_NO_ERROR`	ECI_NO_ERROR
`public final static int ECI_NO_EXTEND`	ECI_NO_EXTEND
`public final static int ECI_SERVERSTATE_DOWN`	ECI_SERVERSTATE_DOWN
`public final static int ECI_SERVERSTATE_UNKNOWN`	ECI_SERVERSTATE_UNKNOWN
`public final static int ECI_SERVERSTATE_UP`	ECI_SERVERSTATE_UP
`public final static int ECI_STATE_ASYNC`	ECI_STATE_ASYNC (2)
`public final static int ECI_STATE_ASYNC_JAVA`	Asynchronous status information call (2)

Table 12. Constants of the ECIRequest class (continued)

Java declaration	Meaning
`public final static int ECI_STATE_CANCEL`	ECI_STATE_CANCEL (3)
`public final static int ECI_STATE_CHANGED`	ECI_STATE_CHANGED (3)
`public final static int ECI_STATE_IMMEDIATE`	ECI_STATE_IMMEDIATE
`public final static int ECI_STATE_SYNC`	ECI_STATE_SYNC (4)
`public final static int ECI_STATE_SYNC_JAVA`	Synchronous status information call (4)
`public final static int ECI_STATUS_LENGTH`	Length of the ECI status block
`public final static int ECI_SYNC`	ECI_SYNC (5)
`public final static int ECI_SYNC_TPN`	ECI_SYNC (6)

Notes:

1. ECI_EXTENDED is not supported by the CICS Transaction Gateway for OS/390.
2. For asynchronous status information calls you should use call type `ECI_STATE_ASYNC_JAVA` rather than `ECI_STATE_ASYNC`.
3. `ECI_STATE_CANCEL` and `ECI_STATE_CHANGED` are not supported by any CICS Transaction Gateway.
4. For synchronous status information calls you should use call type `ECI_STATE_SYNC_JAVA` rather than `ECI_STATE_SYNC`.
5. In this case the variable `Transid` in the `ECIRequest` object is used as **eci_transid**.
6. In this case the variable `Transid` in the `ECIRequest` object is used as **eci_tpn**.

Constructors

Several constructors for `ECIRequest` objects are provided. They are based on the following declaration:

```
public ECIRequest(int iCallType
                  String strServer
                  String strUserid
                  String strPassword
                  String strProgram
                  String strTransid
                  byte abitCommarea []
                  int iCommareaLength
                  int iExtendMode
                  int iLuwToken
                  int iMessageQualifier
                  Callbackable calBack
                 )
```

Table 13 shows the public variables set by the parameters of the constructors.

Table 13. Variables set by the ECIRequest constructors

Parameter to the call	Variable set
`iCallType`	`Call_Type`
`strServer`	`Server`
`strUserid`	`Userid`
`strPassword`	`Password`
`strProgram`	`Program`

Table 13. Variables set by the ECIRequest constructors (continued)

Parameter to the call	Variable set
strTransid	Transid
strCommarea	Commarea
iCommareaLength	Commarea_Length
iExtendMode	Extend_Mode
iLuwToken	Luw_Token
iMessageQualifier	Message_Qualifier
calBack	no public variable

The default constructor for ECIRequest

The default constructor leaves all the variables with their default values. You should not pass this ECIRequest object to a CICS Transaction Gateway without setting some of the variables explicitly.

The following code fragment shows how you can use the default constructor for an ECIRequest object eciRequest.

```
ECIRequest eciRequest = new ECIRequest();
```

A basic constructor for ECIRequest

This constructor takes the most commonly used ECI parameters and sets the contents of the object.

The following code fragment shows how you can use the basic constructor for an ECIRequest object eciRequest. The constructed ECIRequest object is for an ECI_SYNC call to CICS server gateway8. The program to be called is BACKEND. There is no user ID, no password, no transaction ID, and no communication area for this request.

```
ECIRequest eciRequest = new ECIRequest(ECIRequest.ECI_SYNC,
                                       "gateway8",
                                       null,
                                       null,
                                       "BACKEND",
                                       null
                                      );
```

An extended constructor for ECIRequest

This constructor extends the basic constructor to include the ECI requests that create extended logical units of work.

The following code fragment shows how you can use the extended constructor for an ECIRequest object eciRequest. The constructed ECIRequest object is for an ECI_SYNC call to CICS server gateway8. The program to be called is BACKEND. There is no user ID, no password, and no transaction ID. A communication area mycommarea is supplied, and it is 132 bytes long. This request will start a new logical unit of work.

```
ECIRequest eciRequest = new ECIRequest(ECIRequest.ECI_SYNC,
                                       "gateway8",
                                       null,
                                       null,
                                       "BACKEND",
                                       null,
```

```
                                        mycommarea,
                                        132,
                                        ECIRequest.ECI_EXTENDED,
                                        ECIRequest.ECI_LUW_NEW
                                       );
```

A full constructor for ECIRequest

This constructor includes a parameter to set a callback routine.

The following code fragment shows how you can use the full constructor for an `ECIRequest` object `eciRequest`. The constructed `ECIRequest` object is for an ECI_ASYNC call to CICS server `gateway8`. The program to be called is BACKEND. There is no user ID, no password, no transaction ID, and no communication area. The request will be a single unit of work. When the request completes, the `Callbackable` object `mycallback` will be called, and it will be passed the message qualifier `myMQ` to identify the request that is completing.

```
ECIRequest eciRequest = new ECIRequest(ECIRequest.ECI_SYNC,
                                        "gateway8",
                                        null,
                                        null,
                                        "BACKEND",
                                        null,
                                        null,
                                        null,
                                        ECIRequest.ECI_NO_EXTEND,
                                        ECIRequest.ECI_LUW_NEW
                                        myMQ
                                        mycallback
                                       );
```

Methods

The following methods are described:

- `getCICSRc`
- `getRc`
- `getStatus` (synchronous)
- `getStatus` (asynchronous)
- `listSystems`
- `setCallback`
- `stringClientStatus`
- `stringConnectionType`
- `stringServerStatus`

getCICSRc

```
public int getCICSRc()
```

Retrieves the value of `Cics_Rc` from an `ECIRequest` object.

Parameters

None

Returns

The value of `Cics_Rc`

Throws

Nothing

The following code fragment shows how you can use the getCICSRc method to print the ECI return code from a request represented by eciRequest.

```
System.out.println("ECI return code was "+eciRequest.getCICSRc());
```

getRc

```
public int getRc()
```

Retrieves the return code from an ECIRequest object.

Parameters

None

Returns

The value of the return code from an instance of an ECIRequest object. The possible values are the constants of the GatewayRequest class. “Constants” on page 252 lists the values and explains their meanings.

Throws

Nothing

The following code fragment shows how you can use the getRc method to print the return code from a request represented by eciRequest.

```
System.out.println("Return code was "+eciRequest.getRc());
```

getStatus—synchronous

```
public ECIRequest getStatus(String strServer)
```

Creates an instance of an ECIRequest with call type ECI_STATE_SYNC_JAVA.

Parameters

strServer

Optional parameter. The name of the CICS server (**eci_system_name**) to which an instance of an ECIRequest object is to be directed.

Returns

An ECIRequest object that represents a **CICS_ExternalCall** with **eci_call_type** ECI_STATE_SYNC_JAVA.

Throws

Nothing

The following code fragment shows how you can use the getStatus method for an ECIRequest object eciRequest to create an ECIRequest object g8Status to enquire about the status of server gateway8.

```
ECIRequest g8Status = eciRequest.getStatus("gateway8");
```

getStatus—asychronous

```
public ECIRequest getStatus(String strServer
                            int iMessageQualifier
                            Callbackable calBack
                           )
```

Creates an instance of an ECIRequest with call type ECI_STATE_ASYNC_JAVA and a callback routine.

Parameters

strServer
Optional parameter. The name of the CICS server (**eci_system_name**) to which an instance of an `ECIRequest` object is to be directed.

iMessageQualifier
Optional parameter. The identifier for this asynchronous status enquiry.

calBack
A `Callbackable` object that represents the callback routine.

Returns
An `ECIRequest` object that represents a **CICS_ExternalCall** with **eci_call_type** `ECI_STATE_ASYNC_JAVA`.

Throws
Nothing

The following code fragment shows how you can use the `getStatus` method of an `ECIRequest` object `eciRequest` to set up a request that represents a **CICS_ExternalCall** with call type `ECI_STATE_SYNC_JAVA`. The request includes a message identifier of 999, and a callback routine, and is directed at the default server.

```
ECIRequest g8Status = eciRequest.getStatus("",
                                           999,
                                           mycallback
                                          );
```

listSystems

```
public static ECIRequest listSystems(int iNumOfSys)
```

Creates an `ECIRequest` object that corresponds to the **CICS_EciListSystems** call described in *CICS Family: Client/Server Programming* .

Parameters

iNumOfSys
The number of systems to be listed.

Returns
An `ECIRequest` object that represents a **CICS_EciListSystems** request.

Throws
Nothing

The following code fragment shows how you can use the `listSystems` method to construct an `ECIRequest` object `eciRequest` that represents a **CICS_EciListSystems** request for up to ten servers.

```
ECIRequest eciRequest = ECIRequest.listSystems(10);
```

setCallback

```
public void setCallback(Callbackable calBack)
```

Sets a callback routine in an `ECIRequest` object.

Parameters

calBack
A `Callbackable` object that represents the callback routine.

Returns
Nothing

Throws
Nothing

The following code fragment shows how you can use the `setCallback` method of an `ECIRequest` object `eciRequest` . The effect is to set the `Callbackable` object `mycallback` as the callback routine for asynchronous requests for `eciRequest`.

```
eciRequest.setCallback(mycallback);
```

stringClientStatus

```
public String stringClientStatus(int iClientStatus)
```

Converts the value of the `CicsClientStatus` variable of an `ECIRequest` object into a string.

Parameters

iClientStatus
The client status to be interpreted. The client status should be an instance of the `CicsClientStatus` variable in an `ECIRequest` object that has been used in a call with call type `ECI_STATE_SYNC_JAVA` or `ECI_STATE_ASYNC_JAVA`.

Returns
A string describing the client status in words.

Throws
Nothing

The following code fragment shows how you can use the `stringClientStatus` method of an `ECIRequest` object `g8status` to display the status of a client.

```
System.out.println("Client status: "+
        g8status.stringClientStatus(g8status.CicsClientStatus);
```

stringConnectionType

```
public String stringConnectionType(int iConnectionType)
```

Converts the value of the `ConnectionType` variable of an `ECIRequest` object into a string.

Parameters

iConnectionType
The connection type to be interpreted. The connection type should be an instance of the `ConnectionType` variable in an `ECIRequest` object that has been used in a call with call type `ECI_STATE_SYNC_JAVA` or `ECI_STATE_ASYNC_JAVA`.

Returns
A string describing the connection type in words.

Throws
Nothing

The following code fragment shows how you can use the `stringConnectionType` method of an `ECIRequest` object `g8status` to display the type of a connection.

```
System.out.println("Connection type: "+
        g8status.stringConnectionType(g8status.ConnectionType);
```

stringServerStatus

```
public String stringServerStatus(int iServerStatus)
```

Converts the value of the CicsServerStatus variable of an ECIRequest object into a string.

Parameters

iServerStatus
The server status to be interpreted. The server status should be an instance of the CicsServerStatus variable in an ECIRequest object that has been used in a call with call type ECI_STATE_SYNC_JAVA or ECI_STATE_ASYNC_JAVA.

Returns
A string describing the server status in words.

Throws
Nothing

The following code fragment shows how you can use the stringServerStatus method of an ECIRequest object g8status to display the status of a server.

```
System.out.println("Server status: "+
        g8status.stringServerStatus(g8status.CicsServerStatus);
```

The Callbackable class

public interface Callbackable

extends Object

extends Runnable

The asynchronous model supported by the CICS Transaction Gateway and local CICS client allows the use of callback objects. When an asynchronous call is completed, the user-supplied callback object is called with the results of the asynchronous call, and is then run on its own thread.

This interface defines the method that a Callbackable object must provide. It is derived from the standard Runnable interface with one additional method: setResults. This method is called before the object is run to pass in the results of the asynchronous call.

Methods

The following methods are described:

- setResults

setResults

```
public abstract void setResults(GatewayRequest gatResults)
```

Parameters

gatResults
A GatewayRequest object containing the asynchronous results.

The GatewayRequest class

```
java.lang.object
   |
   +----ibm.cics.jgate.client.GatewayRequest
```

public class `GatewayRequest`

extends `Object`

This is the root class for all the different types of request. A user program cannot create a `GatewayRequest` object.

Constants

The `GatewayRequest` object has the following constants, which represent return codes from the attempt to communicate with the CICS Transaction Gateway or local CICS Client.

ERROR_CONNECTION_FAILED

```
public final static int ERROR_CONNECTION_FAILED
```

There was an error in the connection to the CICS Transaction Gateway or local CICS Client.

ERROR_UNKNOWN_REQUEST_TYPE

```
public final static int ERROR_UNKNOWN_REQUEST_TYPE
```

The CICS Transaction Gateway or local CICS Client did not recognize the request type.

ERROR_REPLY_MISMATCH

```
public final static int ERROR_REPY_MISMATCH
```

The reply from the CICS Transaction Gateway or local CICS client did not match the original request type.

ERROR_GATEWAY_CLOSED

```
public final static int ERROR_GATEWAY_CLOSED
```

The CICS Transaction Gateway or local CICS Client has closed.

ERROR_WORK_WAS_REFUSED

```
public final static int ERROR_WORK_WAS_REFUSED
```

The CICS Transaction Gateway or local CICS client refused the request.

The CicsCpRequest class

```
java.lang.Object
   |
   +----ibm.cics.jgate.client.GatewayRequest
           |
           +----ibm.cics.jgate.client.CicsCpRequest
```

public class `CicsCpRequest`

extends `GatewayRequest`

This class allows you to get details of the code page used by a CICS Client. You might need this information in order to set up a communication area for use with an `ECIRequest` object, since the contents of the communication area must be in the code page of the CICS Client. To get the information, you use the `flow` method of a previously defined `JavaGateway` object to pass the `CicsCpRequest` object to the CICS Transaction Gateway, or to the local CICS Client. The returned code page information might be in a workstation-dependent format, or in a Java-encoded format. It can be used in either format with some of the classes provided in JDK 1.1 to convert the data.

You must set up the CICS Client with a valid code page defined. See your client publications for details.

Public variables

Objects of this class have no public variables.

Constants

Table 14 shows the constants of the `CicsCpRequest` class.

Table 14. Constants of the CicsCpRequest class

Java declaration	Meaning
`public final static int CODEPAGE_ERR`	An error was detected.
`public final static int CODEPAGE_NORMAL`	The previous request completed normally.
`public final static int CODEPAGE_NOT_AVAILABLE`	A code page cannot be obtained.
`public final static int NLS_NOT_SUPPORTED`	The Java Virtual Machine does not support this function.
`public final static int NO_ENCODING_CLASS`	(Not used.)
`public final static int OPSYS_NOT_RECOGNISED`	The operating system name (opsys.name) is not recognized.
`public final static int OPSYS_OS390`	The operating system name is OS/390.

Constructors

There is only one constructor for a `CicsCpRequest` object:

```
public CicsCpRequest()
```

This constructor creates a `CicsCpRequest` object.

Parameters

None.

The following code fragment shows the creation of a `CicsCpRequest` object `cpr`.

```
CicsCpRequest cpr = new CicsCpRequest();
```

Methods

The following methods are described:

- `getClientCp`

- getClientRc
- getClientRcString
- getRc

These methods retrieve values from a CicsCpRequest object. The values relate to the most recent use of the object in the flow method of a JavaGateway object.

getClientCp

```
public String getClientCp()
```

Retrieves the value of the client's code page from a CicsCpRequest object.

Parameters
None

Returns
A string representing the client's code page, or unknown.

Throws
Nothing

getClientRc

```
public int getClientRc()
```

Retrieves the client return code from a CicsCpRequest object.

Parameters
None

Returns
The value of the client return code from an instance of an CicsCpRequest object. The possible values are the constants of the CicsCpRequest class.

Throws
Nothing

The following code fragment shows how you can use the getClientRc method to print the return code from a request represented by cpr.

```
System.out.println("Client return code was "+cpr.getClientRc());
```

getClientRcString

```
public String getClientRcString()
```

Retrieves the string representation of the client return code from a CicsCpRequest object.

Parameters
None

Returns
The string value of the client return code from an instance of an CicsCpRequest object. The possible values are the names of the constants of the CicsCpRequest class, or Return Code Out Of Range.

Throws
Nothing

The following code fragment shows how you can use the getClientRcString method to print the return code from a request represented by cpr.

```
System.out.println("Client return code was "+cpr.getClientRcString());
```

getRc

```
public int getRc()
```

Retrieves the return code from a `CicsCpRequest` object.

Parameters

None

Returns

The value of the return code from an instance of an `CicsCpRequest` object. The possible values are the constants of the `GatewayRequest` class. “Constants” on page 252 lists these values and explains their meanings.

Throws

Nothing

The following code fragment shows how you can use the `getRc` method to print the return code from a request represented by `cpr`.

```
System.out.println("Return code was "+cpr.getRc());
```

Appendix J. HTML coded character sets

Table 15 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 15. Coded character sets

Language	Coded character set	IANA charset	IBM CCSID
Albanian	ISO/IEC 8859-1	iso-8859-1	819
Arabic	ISO/IEC 8859-6	iso-8859-6	1089
Bulgarian	Windows 1251	windows-1251	1251
Byelorussian	Windows 1251	windows-1251	1251
Catalan	ISO/IEC 8859-1	iso-8859-1	819
Chinese (simplified)	GB	gb2312	1381 or 5477
Chinese (traditional)	Big 5	big5	950
Croatian	ISO/IEC 8859-2	iso-8859-2	912
Czech	ISO/IEC 8859-2	iso-8859-2	912
Danish	ISO/IEC 8859-1	iso-8859-1	819
Dutch	ISO/IEC 8859-1	iso-8859-1	819
English	ISO/IEC 8859-1	iso-8859-1	819
Estonian	ISO/IEC 8859-1	iso-8859-1	819
Finnish	ISO/IEC 8859-1	iso-8859-1	819
French	ISO/IEC 8859-1	iso-8859-1	819
German	ISO/IEC 8859-1	iso-8859-1	819
Greek	ISO/IEC 8859-7	iso-8859-7	813
Hebrew	ISO/IEC 8859-8	iso-8859-8	916
Hungarian	ISO/IEC 8859-2	iso-8859-2	912
Italian	ISO/IEC 8859-1	iso-8859-1	819
Japanese	Shift JIS EUC Japanese	x-sjis or shift-jis euc-jp	943 (932, a subset of 943, is also valid) 5050 (EUC)
Korean	EUC Korean	euc-kr	970 (for AIX or Unix)
Latvian	Windows 1257	windows-1257	1257
Lithuanian	Windows 1257	windows-1257	1257
Macedonian	Windows 1257	windows-1257	1251
Norwegian	ISO/IEC 8859-1	iso-8859-1	819
Polish	ISO/IEC 8859-2	iso-8859-2	912
Portuguese	ISO/IEC 8859-1	iso-8859-1	819

Table 15. Coded character sets (continued)

Language	Coded character set	IANA charset	IBM CCSID
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)

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

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: 5655-147

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5445-00

Spine information:

CICS TS for OS/390 CICS Internet Guide Release 3