Workload Management for Web Access to CICS

Network Dispatcher, Sysplex Distributor, DNS connection optimization, port sharing

Load balance using CICS dynamic routing and CICSPlex SM

Covers CICS Web support, CICS Transaction Gateway, CICS CORBA, and HOD

Phil Wakelin
Steve Burghard
Jeff Albright
Cavin Edwards
Peter Klein

ibm.com/redbooks
Workload Management for Web Access to CICS

February 2001
Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special notices” on page 223.

First Edition (February 2001)

This edition applies to:

- Version 2.10 of OS/390 (program number 5647-A01)
- Version 1.3 of CICS Transaction Server for OS/390 (program number 5655-147)
- Version 3.12 of CICS Transaction Gateway (program number 5648-B43)
- Version 3.12 of IBM CICS Universal Clients (program number: 5648-B42)
- Version 4 of IBM SecureWay Host On-Demand (program number 5648-C54)
- Version 1 of WebSphere Edge Server (program number 5648-D77)
- Version 3.02 of WebSphere Application Server, Standard Edition (program number 5801-AAR)
- Version 3.02 of WebSphere Application Server for OS/390 (program number 5655-A98).

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE  Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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

© Copyright International Business Machines Corporation 2001. All rights reserved.
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
## Contents

**Figures** .......................................................... vii

**Tables** ............................................................ xiii

**Preface** ........................................................... xv

The team that wrote this redbook ................................ xvii

Comments welcome .................................................. xxi

---

**Part 1. Basic training — CICS Web-enablement and WLM** .................. 1

Chapter 1. **CICS and Web-enabling** ...................................... 3

1.1 The separation of presentation and business logic ............... 4

1.2 CICS Web-enabling technologies ................................... 6

1.2.1 CICS Web support .......................................... 6

1.2.2 CICS Transaction Gateway .................................. 14

1.2.3 CICS CORBA client support .................................. 24

1.2.4 Host On-Demand ............................................ 26

1.3 CICS Web selection table ........................................... 30

Chapter 2. **Workload management technologies** ....................... 31

2.1 MVS Workload Manager ........................................... 32

2.2 Communications Server ........................................... 33

2.2.1 Port sharing ................................................ 33

2.2.2 DNS connection optimization ................................ 34

2.2.3 VIPA ...................................................... 36

2.2.4 Sysplex Distributor ......................................... 37

2.2.5 VTAM generic resource ....................................... 38

2.3 CICS Transaction Server .......................................... 39

2.3.1 CICSPlex SM ................................................ 39

2.3.2 Dynamic routing ............................................. 41

2.3.3 VSAM RLS .................................................. 41

2.4 CICS Transaction Gateway ........................................ 42

2.4.1 CICS Universal Client Workload Manager .................... 42

2.4.2 EXCI ........................................................ 42

2.5 WebSphere Application Server .................................... 43

2.5.1 WebSphere on distributed platforms ........................ 43

2.5.2 WebSphere on OS/390 ....................................... 44

2.6 Network Dispatcher .............................................. 45

2.6.1 Dispatcher .................................................. 45

2.6.2 Content Based Routing ..................................... 47

2.6.3 Interactive Session Support .................................. 47
Part 3. The construction zone — building the solution

Chapter 7. Workload management for HOD

7.1 Web server ............................................ 100
7.2 Telnet server ........................................ 100
  7.2.1 DNS connection optimization ....................... 100
  7.2.2 Service Location Protocol (SLP) .................. 103
7.3 CICS 3270 access .................................... 106

Chapter 8. CICS Web support ................................ 111
8.1 Configuring the scenario ................................ 112
  8.1.1 Configuring TCP/IP .................................. 113
  8.1.2 Configuring CICS .................................... 121
  8.1.3 Configuring CICSPlex SM ............................ 128
8.2 Running our workload .................................. 131
  8.2.1 Failover scenarios .................................. 134

Chapter 9. CTG servlets with WebSphere AS (OS/390) ... 141
9.1 Configuring the scenario ................................ 142
  9.1.1 Configuring TCP/IP .................................. 143
  9.1.2 Configuring WebSphere AS and the CTG .......... 144
  9.1.3 Configuring CICS ..................................... 150
  9.1.4 Configuring CICSPlex SM ............................ 154
9.2 Running our workload .................................. 164
  9.2.1 Failover scenarios .................................. 168

Chapter 10. CTG servlets with WebSphere AS (Windows NT) ... 173
10.1 Configuring the scenario ............................... 174
  10.1.1 Configuring Network Dispatcher ................. 175
  10.1.2 Configuring WebSphere AS and the CTG ....... 181
  10.1.3 CICS configuration .................................. 195
10.2 Running our workload .................................. 196
  10.2.1 Failover scenarios .................................. 202

Appendix A. The Trader application .......................... 205
A.1 3270 Trader ............................................. 205
A.2 Trader via CWS and a Web-aware converter .......... 210
A.3 Trader via a servlet and the CTG ...................... 215

Appendix B. Using the additional material ..................... 221
B.1 Locating the Web material .............................. 221
B.2 Using the Web material ................................. 221
Figures

1. Separation of CICS business and presentation logic .................... 4
2. CICS Web support overview ........................................... 6
3. CICS Web support — direct connection .................................. 8
4. CICS Web support — CICS WebServer Plugin ............................ 10
5. CICS Web support — 3270 Web bridge .................................. 12
6. CICS Transaction Gateway overview ..................................... 14
7. CTG servlet architecture on a distributed platform ................. 18
8. Servlet using a remote CTG on OS/390 .................................. 19
9. CTG servlet architecture on OS/390 ................................. 20
10. CTG applet architecture on a distributed platform ............ 22
11. CTG applet architecture on OS/390 ................................... 23
12. CICS CORBA client support overview .................................. 24
13. Components in CICS CORBA client support ......................... 25
14. Host On-Demand overview ............................................. 26
15. DNS connection optimization and TCP/IP port sharing ........... 35
16. CICS pseudo-conversational processing .............................. 49
17. Web server state — showing state tokens .............................. 51
18. HTML URL rewriting .................................................... 52
19. A sample HTTP cookie ................................................... 53
20. Hidden HTML form field .................................................. 54
21. Shared temporary storage, for storing Web state data ........... 56
22. Session based state data 1 ............................................. 58
23. Session based state data 2 ............................................. 59
24. CWS Web-aware workload management options .................. 65
25. TCP/IP port sharing ..................................................... 66
26. TCP/IP workload management in a sysplex .......................... 67
27. TCP/IP port sharing — OS/390 Web server ......................... 68
28. CWS and dynamic routing ............................................... 71
29. Types of SSL handshake ................................................. 72
30. CWS and SSL sessions — caching of session-ID .................... 73
31. State management for 3270 Web bridge ............................... 75
32. CWS 3270 Web bridge workload management overview .......... 76
33. CTG OS/390 workload management options ........................ 79
34. TCP/IP port sharing ..................................................... 80
35. TCP/IP port sharing ..................................................... 81
36. DFHXCURM and workload balancing .................................. 84
37. MRO and dynamic routing ............................................... 85
38. Network Dispatcher and WebSphere AS .............................. 86
39. Network Dispatcher and Java gateway application ............... 87
40. CICS Universal Client Workload Manager ............................ 88
41. MRO and dynamic routing ............................................. 89
42. CICS CORBA client support ........................................... 91
43. TCP/IP port sharing ................................................... 92
44. TCP/IP workload management in a sysplex ......................... 93
45. CICS CORBA client support and dynamic routing. ............... 94
46. CICS Native IIOP support: the role of the REQUESTMODEL ....... 95
47. HOD access to CICS .................................................... 99
48. DNS connection optimization for TN3270 servers ................. 101
49. Host On-Demand recovery with DNS connection optimization .... 103
50. HOD workload balancing with SLP .................................. 105
52. CICS workload management. ......................................... 106
53. CICS Web support scenario ......................................... 111
54. CWS Trader application .............................................. 112
55. DNS connection optimization configuration ....................... 113
56. TCP/IP stack profile member showing SYSPLEXROUTING ........ 114
57. NAMED port allocation ............................................... 114
58. AUTOLOG for NAMED procedure .................................... 114
59. TCPDATA member: NSINTERADDR statement ..................... 115
60. Local name server procedure, NAMED ............................ 115
61. named.boot file ..................................................... 116
62. Forward domain data file: named.for ............................. 117
63. Reverse domain data file: named.rev ............................. 117
64. Hints file: named.ca .................................................. 118
65. Loopback file: named.lbk ........................................... 118
66. PING command output .............................................. 119
67. CICS message for DNS connection optimization .................. 119
68. TCP/IP SHAREPORT statement ................................... 120
69. TRADERBL program definition — screen 1 ....................... 122
70. TRADERBL program definition — screen 2 ....................... 122
71. RLS file definition for file CUSTFILE ............................... 123
72. RLS file definition for file COMPFILE ............................ 123
73. DOCTEMPLATE definition. ........................................... 124
74. DNS.TRAD, TCPIPSERVICE definition ............................. 125
75. Temporary storage model definition ................................ 127
76. CICSPlex configuration .............................................. 128
77. CICSPlex SM — PAA1PAA4 CICSGRP ............................. 129
78. CICSPlex SM — TRADER WLMSPEC ............................. 129
79. ‘CICSPlex SM — Add Scope for WLMSPEC ....................... 130
80. CICS Web support — Trader workload ............................ 131
81. NETSTAT confirmation of DNS connection optimization .......... 132
82. CICSPlex SM WUI display — usage count for TRADERCV ........ 133
83. CICSPlex SM WUI display — usage count for TRADERBL ........ 133
84. CICS statistics — TCP/IP services ........................................... 134
85. CWS — quiesce of a listener region ...................................... 135
86. CWS — cancel of a listener region ....................................... 136
87. CWS — closure of a TCPIPSERVICE ...................................... 137
88. CWS — quiesce of an AOR ................................................... 138
89. CWS — cancel of an AOR ................................................... 139
90. CWS — restart of an AOR ................................................... 140
91. WebSphere OS/390 scenario .................................................. 141
92. Trader servlet application .................................................... 142
93. SHAREPORT statement, for the Web servers ......................... 143
94. TCP/IP display of Web server shared port 99 ........................... 143
95. Web server startup JCL ....................................................... 144
96. Configuration of OS/390 was.conf file .................................... 145
97. OS/390 CTG Trader application ............................................. 146
98. WebSphere classpath and libpath for OS/390. ......................... 147
99. Sample DFHXCURM logic .................................................... 148
100. Program definition for TRADERB2 ....................................... 150
101. DHFCNV entry for program TRADERB2 ................................ 151
102. Transaction definition for mirror TRB2 .................................. 152
103. EXCI connection definition CTG2 ...................................... 153
104. EXCI sessions definition EXCGSESS ................................... 153
105. CICSPlex configuration ..................................................... 154
106. CICSPlex SM — TRADER WLMSPEC ................................... 155
107. CICSPlex SM — Create TRANGRP ...................................... 156
108. CICSPlex SM — WLMDEF TRB2 ......................................... 156
109. CICSPlex SM — Install WLMDEF ........................................ 157
110. MVS WLM — Initial welcome screen .................................... 157
111. MVS WLM — choose service definition .................................. 158
112. MVS WLM — definition menu .............................................. 158
113. MVS WLM — service class selection .................................... 159
114. MVS WLM — create a service class’ ..................................... 159
115. MVS WLM — choose a goal type ......................................... 160
116. MVS WLM — average response time goal .............................. 160
117. MVS WLM — response times defined .................................... 161
118. MVS WLM — service class TRAD created .............................. 161
119. MVS WLM — add TRB2 to service class TRAD (1) .................... 162
120. MVS WLM — add TRB2 to service class TRAD (2) .................... 162
121. MVS WLM — add TRB2 to service class TRAD (3) .................... 163
122. MVS WLM — add TRB2 to service class TRAD (4) .................... 163
123. The OS/390 Trader servlet workload .................................... 164
124. Activity statistics for Web server WEBCICS1 ........................ 166
125. Activity statistics for Web server WEBCICS2 ........................ 166
126. CICSPlex SM WUI display — usage count for TRADERBL .......... 167
127. WebSphere OS/390 — cancel of a Web server .......................... 168
128. WebSphere OS/390 — quiesce of a listener region .................. 169
129. WebSphere OS/390 — cancel of a listener region .................. 170
130. WebSphere OS/390 — restart of a listener region .................. 171
131. WebSphere and CTG on Windows NT scenario ...................... 173
132. Trader servlet application ........................................... 174
133. The Network Dispatcher topology .................................... 175
134. Network Dispatcher — defining a cluster .......................... 175
135. Network Dispatcher — adding a port ................................ 176
136. Network Dispatcher — adding a server ............................. 176
137. Configuring the MS loopback adaptor ................................ 177
138. Network Dispatcher — inspecting the routing table ............... 178
139. Connecting to the Dispatcher host machine ......................... 179
140. Network Dispatcher — modifying the weights ...................... 180
141. Windows NT CTG servlet application ................................ 181
142. WebSphere — task wizard 1 .......................................... 182
143. WebSphere — task wizard 2 .......................................... 183
144. WebSphere — task wizard 3 .......................................... 184
145. WebSphere — task wizard 4 .......................................... 185
146. WebSphere — task wizard 5 .......................................... 186
147. WebSphere — setting the classpath .................................. 187
148. WebSphere — adding the PATH environment variable ............ 188
149. WebSphere — adding aliases ......................................... 189
150. CTG configuration tool — server definition ......................... 191
151. CTG configuration tool — protocol ................................... 191
152. CTG configuration tool — SNA definition ............................ 192
153. CTG configuration tool — Server connections ....................... 192
154. CTG configuration tool — Workload Manager settings .............. 193
155. CTG configuration tool — adding a program ......................... 194
156. CTG configuration tool — program definitions ...................... 194
157. CBPS sessions definition ............................................. 195
158. VTAM APPL definition .............................................. 195
159. WebSphere NT — Trader workload ................................... 196
160. Network Dispatcher server weights .................................. 197
161. Network Dispatcher — active connections .......................... 198
162. WebSphere — performance analyzer .................................. 198
163. Personal Communications — active sessions ........................ 199
164. CICSPlex SM WUI — DFHCNV calls before workload .............. 200
165. CICSPlex SM WUI — DFHCNV calls after workload ............... 200
166. CICSPlex SM WUI — TRADERBL calls ............................. 201
167. WebSphere NT — stopping a Web server ........................... 202
168. WebSphere NT — cancel of a Web server ........................... 203
169. WebSphere NT — restart of failed listener region ............... 204
Tables

1. CICS Web selection table .................................................30
2. Summary of techniques for managing state data ....................61
3. EXCI retryable responses ..............................................83
4. CICS resources used by the Trader application ..................121
5. CICSplex SM resources ...............................................129
6. CWS results — quiesce of a listener region .......................135
7. CWS results — cancel of a listener region .........................136
8. CWS results — closure of a TCPIPSERVICE .......................137
9. CWS results — quiesce of an AOR .................................138
10. CWS results — cancel of an AOR .................................139
11. CWS results — restart of an AOR .................................140
12. CICS resources used by the Trader servlet application .........150
13. CICSplex SM resources — Goal mode ............................155
14. WebSphere OS/390 results — cancel of a Web server ..........168
15. WebSphere OS/390 — quiesce of a listener region .............169
16. WebSphere OS/390 results — cancel of a listener region .......170
17. WebSphere OS/390 results — restart of a listener region ......171
18. WebSphere NT scenario — CICS program usage .................201
19. WebSphere NT results — stopping a Web server ...............202
20. WebSphere NT results — cancel of a Web server ...............203
21. WebSphere NT results — restart of failed listener region ......204
Preface

The Internet seems all-pervasive, and now it is in your CICS systems. This means, not just more work for you, but also more work for your CICS systems and your I/T infrastructure. How can you design your systems to handle it all?

In this IBM Redbook we first provide an introduction to Web-enabling CICS, followed by a brief overview of the building-blocks that provide the backbone of IBM’s workload management technologies. Included in this discussion are technologies such as DNS connection optimization, TCP/IP port sharing, and Sysplex Distributor (as provided by IBM OS/390 Communications Server); together with Network Dispatcher, CICSPlex SM and CICS dynamic routing, the CICS Universal Client workload manager, and VSAM Record Level Sharing. Following this, we explain how to use these components to design a load balancing solution for whichever CICS Web-enablement strategy you have chosen.

Finally, we document three real life scenarios, where we designed, built, and tested three load balanced CICS Web-enabled solutions — a Web-aware CICS Web support application, and two servlet solutions using the CICS Transaction Gateway from both WebSphere Application Server for OS/390 and from WebSphere Application Server for Windows NT.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

Phil Wakelin is a Senior I/T Specialist at the International Technical Support Organization, San Jose Center, and has 10 years experience working on most platforms and versions of CICS. He writes extensively and has taught IBM classes on many areas of CICS, specializing in the CICS Web-enablement and client-server technology. Before joining the ITSO, Phil worked in the Installation Support Center, IBM UK as a pre-sales support specialist for CICS client-server, and prior to this he worked in the CICS System Test department of IBM Hursley.

Steve Burghard is a CICS Technical Solutions Specialist from IBM UK, based in North Harbour, Portsmouth. He has 18 years experience of CICS and has spent the last 5 years with IBM, mostly supporting the IBM internal accounts. He has contributed widely to other ITSO CICS Redbook publications, and has a vast range of experience in all areas of CICSPlex System Manager and CICS Transaction Server for OS/390.
Jeff Albright is a Consulting I/T Specialist with Albright Consulting Services, an IBM Business Partner based in Charlotte, North Carolina. He has 22 years of experience in IBM large systems. His areas of expertise include workload and performance management for OS/390, CICS, and DB2 as well as development of Internet and intranet infrastructures. He is currently working on various performance, high availability, and Web deployment projects.

Cavin Edwards is a Solution Test Specialist from IBM UK, based in Hursley. He has 7 years of experience in Object Oriented and Client/Server computing, in all areas of the software life cycle, using languages and tools such as C++, Java and Delphi, within OS/390, UNIX and PC environments. For the last year he has worked in the CICS Solution Test Group, with Java and Web based software for a variety of test and customer focused solutions.

Peter Klein is a CICS team leader at the CICS second-level support for IBM Germany. He has 16 years of experience on IBM software products working as a technical support specialist, 14 of those devoted to CICS. His expertise includes CICSPlex SM, CICS language support and distributed CICS systems. He has contributed to other ITSO CICS Redbook publications.

Thanks to the following people for their invaluable contributions to this project:

Geoff Sharman, Dave Rushall, Greg Thurtle, John Tiling, Paul Johnson, Paul Harris, Kate Robinson, Steve Wallin, Mark Cocker, George Burgess, Grant Shayler, Chris Smith, Noel O’Dowd
IBM Hursley, UK.

Nigel Williams, Steve Wall, Fabrice Jarassat
IBM Design Centre for e-transaction processing, Montpellier, France

Carol Shanesy, Leigh Compton, Dennis Weiand, George Hoyer, Carl Moje
IBM Dallas

Rich Conway, Bob Haimowitz, Frank Kyne, of the International Technical Support Organization

Michael E. Baskey
IBM Poughkeepsie

Chris Gage, Richard Nesbitt, Jay Aiken, Mac Devine
IBM Raleigh
Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 245 to the fax number shown on the form.
- Use the online evaluation form found at ibm.com/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com
Part 1. Basic training — CICS Web-enablement and WLM

In this part we give an overview of the four strategic CICS Web-enablement technologies (CICS Web support, CICS Transaction Gateway, CICS CORBA client support, and Host On-Demand).

We then provide a brief introduction to all the different workload balancing technologies that can be used in conjunction with Web access to CICS, together with reference information on where to find more details.

Finally, we introduce the subject of state management, and discuss the different solutions that are available to manage state in Web applications.
Chapter 1. CICS and Web-enabling

In this chapter we provide an introduction to the strategic CICS Web-enabling technologies covered in this book, together with reference information. If you are already familiar with these technologies, you may want to refer to the introductory material in Chapter 2, “Workload management technologies” on page 31. Or, you can simply skip to Part 2, “The think tank — designing the solution” on page 63 for information on how these technologies can be integrated in a workload management environment.

The four strategic CICS Web-enabling technologies we will discuss are:

- CICS Transaction Gateway (CTG)
- CICS Web support (CWS)
- CICS CORBA client support
- Host On-Demand (HOD)

First, we give a brief overview of the principles of CICS modular application design as it relates to CICS Web-enabling, before presenting an introduction to each of the CICS Web-enabling technologies. Finally, we offer our “CICS Web selection table”, which compares these four different Web-enabling technologies and helps you choose among them.

A more detailed overview of each of these CICS Web-enabling technologies is given in the redbook Revealed! Architecting Web Access to CICS, SG24-5466, and further information on the CICS Web selection table can be found in the CICS Web selection guide whitepaper, available at:


This book refers to the Web-enabling function in CICS Transaction Server for OS/390 Version 1 Release 3 (CICS TS V1.3), CICS Transaction Gateway (CTG) V3.11, and Host On-Demand (HOD) V4.0. You should be aware that the previous releases of CICS TS, the CTG, and HOD do not necessarily provide the same level of function as we document in this book.
1.1 The separation of presentation and business logic

A sound principle of CICS application design is to separate the presentation logic from the business logic. Communication between the programs is done by using the EXEC CICS LINK command, and data is passed between such programs in a buffer known as a communication area (COMMAREA). The structure of this data in the COMMAREA is also part of the application design. This is illustrated in Figure 1.

![Figure 1. Separation of CICS business and presentation logic](image)

Such a modular design provides not only a separation of functions, but is also the key for re-use of existing applications using different presentation methods (such as the Web accesses that are the subject of this book). These presentation logic and business logic modules are still executed together as a single CICS task, but if designed in this form, then they can also readily exploit the distributed program link (DPL) and workload management functions provided by CICS to spread work within a sysplex, or between different CICS systems distributed across a network. For further details on a modern CICS modular application design, refer to the publication *Designing and Programming CICS Applications*, ISBN 1-56592-676-5, SR23-9692.
The different means by which the business logic in a CICS application can be invoked using a COMMAREA interface are as follows:

- From a CICS application, where the presentation logic is HTTP-based (Web-aware)
- From the CICS Universal Client, using the External Call Interface (ECI) on a workstation
- From a Java applet, servlet, or application, using the facilities of the CICS Transaction Gateway and the Java version of the external call interface (ECI).
- From a Common Object Request Broker Architecture (CORBA) client, using the IIOP protocol and the JCICS classes
- From another program running in the OS/390 sysplex (such as a Web server GWAPI program), using the EXCI (External CICS Interface) interface
- From any program which uses a EXEC CICS LINK and a COMMAREA structure to pass data

Do not forget that there is a restriction on the size of data that can be passed in a CICS COMMAREA. The maximum size of this area is 32K (or more precisely 32,500 bytes). With CWS in CICS TS V1.3 you now have the choice of using the CICS DOCUMENT and WEB API to send and receive HTTP data streams longer than 32 KB, and with CICS CORBA client support you can also pass messages longer than 32 KB using temporary storage queues.

Note that if existing applications are separated into presentation and business logic modules, you should be aware that a distributed program link (DPL) request (as opposed to a local EXEC CICS LINK) imposes restrictions on some CICS functions that can be used. These are documented in Appendix G of the CICS Application Programming Reference, SC33-1688. Furthermore, some applications make extensive use of terminal features such as the CICS Terminal ID or the TCTUA. Obviously these functions can only be used in the 3270 presentation logic modules and cannot be utilized in the business logic.

Many (if not the majority of) legacy applications were designed or written without a separation of presentation and business logic. Most application programmers did not plan for the Year 2000 data change, never mind Web-enabling their applications. As a result, these applications are often deemed too difficult to re-engineer. For that reason IBM has developed Web-enabling technologies that allow re-use of the 3270 interface, as well as technologies that require a callable COMMAREA interface.
1.2 CICS Web-enabling technologies

In the following sections we give a short introduction to the four strategic CICS Web-enabling technologies discussed in this book.

1.2.1 CICS Web support

CICS Web support (CWS) is a set of resources supplied with CICS TS for OS/390 V1.3 and CICS TS for VSE V1.1 that provide CICS with some functionality similar to a real Web server. A summary of how a CICS application can be Web-enabled using the CWS is illustrated in Figure 2.

![CICS Web support overview](image)

**Figure 2. CICS Web support overview**

---

**CICS Web Interface**

In CICS Transaction Server for OS/390 V1.3, the CICS Web functionality, previously known as the CICS Web Interface (CWI), was split into the listener support for TCP/IP and the protocol support for HTTP, and was also internally redesigned. This book reflects the new structure in CICS TS V1.3, where the TCP/IP listener support is provided by the CICS Sockets domain, and the HTTP protocol support is provided by CICS Web support.
CWS provides a native HTTP interface to CICS; this interface can be used by both 3270 based transactions and applications that provide a callable COMMAREA interface. Two different configurations can be used to route the HTTP requests into the CICS region. Both configurations allow the use of the same facilities in CICS, although the configuration of the two options is significantly different. These configurations are as follows:

- **A direct connection** from a Web browser to CICS. This uses the facilities of the CICS TCP/IP listener to pass the request directly into CICS Web support.

- Through the OS/390 Web server using the facilities of the CICS WebServer Plugin. This is a CICS supplied GWAPI extension to the OS/390 Web server. It routes requests from the OS/390 Web server into the CICS region using the EXCI communication mechanism. The supplied GWAPI module is called DFHWBAPI, and was previously termed the “ICAPI DLL”.

With both, the direct connection and the CICS WebServer Plugin, CWS can be used to invoke two types of CICS applications:

- To invoke a **3270 transaction**, the facilities of the 3270 bridge are used. The 3270 transaction remains unchanged and the 3270 output is converted to HTML. We will refer to this function as the **3270 Web bridge**. This function is only available when using CICS Transaction Server V1.2 or higher.

- To invoke an existing application that provides a callable COMMAREA interface, some new CICS presentation logic must be written. This logic uses CICS facilities to interpret, act upon, and then build and return the HTTP data stream. We will refer to a CICS application containing such logic as “**Web-aware**”. This Web-aware logic can be contained either within the CWS converter Encode and Decode routines, within the original program, or in a separate Web presentation module that links to the original program. To create this Web-aware presentation logic there are two different methods provided by CWS:
  - WEB API
  - COMMAREA manipulation

The WEB API, together with the DOCUMENT API and TCPIP API, provide a rich set of functions to interpret, manipulate, and build the HTTP data streams within a CICS application. They are part of the new function of CWS in CICS TS V1.3, and are described in more detail in Chapter 12 of *CICS Internet Guide*, SC34-5445, and Chapter 3 of *CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge*, SG24-5480.
The COMMAREA manipulation technique was originally introduced with CWI support in CICS/ESA V4.1. It uses the CICS COMMAREA as a buffer for transferring the HTTP data stream along with a range of utility programs to manipulate the data stream. The CWS HTML template manager program (DFHWBTL) is used to build the response. This technique is still available in CICS TS V1.3, but for ease of use and higher functionality, we recommend use of the WEB API.

### 1.2.1.1 CICS Web support — direct connection

Figure 3 illustrates the major components of CICS Web support when using Web-aware presentation logic via a direct connection to CICS. The light shading is for CWS components that run under the Web attach transaction; the darker shading for CWS components that run under the alias transaction.

Using the CWS direct connection allows greater than 32KB of data to be returned from the application to the Web browser, although only 32KB of data can be received by the application.
CICS TCP/IP listener
The CICS TCP/IP listener is part of the new CICS Sockets domain, and runs as the CSOL (Sockets listener) system task. It provides TCP/IP support to handle requests for internal CICS functions that use TCP/IP services, currently HTTP and IIOP support. As such, it is not a component of CWS, but a service used by CWS. The CICS TCP/IP listener is completely separate from, and not to be confused with, the CICS TCP/IP Sockets interface, which provides an application level TCP/IP socket interface to CICS applications. It is described in the book CICS/ESA and TCP/IP for MVS Sockets Interface, GG24-4026.

Web attach transaction
The Web attach transaction (CWXN) performs the Web attach processing. It invokes the DFHCCNV code page conversion routine, links to the analyzer, and then invokes the alias. The CWXN task will terminate after invoking the alias, unless persistent HTTP connections are used.

DFHCCNV
The DFHCCNV code page conversion routines are invoked by the Web attach processing to convert the ASCII HTTP headers and user data of the Web browser client to EBCDIC, and by the alias transaction to convert EBCDIC output back to ASCII.

Analyzer
The purpose of the analyzer is to analyze the incoming HTTP request. It runs under the transaction CWXN, and decides if the request will be accepted, and if so, what parameters will be set. Among other things, it decides the name of the alias, converter, userid, and user program.

Alias
The alias transaction is invoked by the analyzer. The default alias transaction code is CWBA, but this can be modified. The alias initially invokes the program DFHWBA, which links to the business logic interface.

Business logic interface
The business logic interface (BLI) is an externally callable interface to CWS. It is implemented by the module DFHWBBLI. It provides a mechanism for implementing Web-aware presentation logic in the "converter". The converter provides Decode and Encode routines to receive and send the HTTP presentation logic. Note that it is possible to bypass the converter and implement the Web-aware logic in a separate module which would communicate directly with the business logic via a COMMAREA interface.
1.2.1.2 CICS WebServer Plugin
An alternative approach to accessing CICS Web support is through the services of the OS/390 Web server, using the CICS WebServer Plugin, (DFHWBAPI). In this implementation, some of the function previously handled through the CICS-supplied programs for CICS Web support is now replaced by function within the Web server and its plugin.

The OS/390 Web server has been renamed at various times to reflect its positioning within IBM's OS/390 Internet product portfolio. The Internet Connection Secure Server (ICSS) Web server became the Lotus Domino Go Webserver for OS/390. This has now been renamed to WebSphere Application Server for OS/390, which provides the IBM HTTP Server. Whichever Web server you use, we will refer to it as the OS/390 Web server.

The CICS WebServer Plugin replaces the functionality of the CWS Web attach transaction, described previously. The OS/390 Web server has to be configured with a Service directive in order to function with the CICS WebServer Plugin. This configuration is described in the CICS Internet Guide, SC34-5445. Using this service directive, the OS/390 Web server receives the HTTP request, builds an EXCI request, and invokes the BLI using the CSMI mirror transaction in the target CICS region. The HTTP data stream is passed to the BLI in an EXCI COMMAREA.

Figure 4 illustrates the major components of CICS Web support when using Web-aware presentation logic via the CICS WebServer Plugin.

![Figure 4. CICS Web support — CICS WebServer Plugin](image-url)
The same facilities within CICS are available using the CICS WebServer Plugin as when using a direct connection, but there are a few important differences, which are summarized below:

- The OS/390 Web server and the CICS region must be running within the same OS/390 sysplex, since the CICS WebServer Plugin uses the EXCI communication mechanism.
- Only 32 KB of data in the HTTP data stream can be passed to or from the CICS program when using the CICS WebServer Plugin. This is because the EXCI uses a standard CICS COMMAREA on which the restriction of 32 KB applies.
- Security processing is performed in the OS/390 Web server if using the CICS WebServer Plugin.
- Code page conversion is performed in the OS/390 Web server, not in CICS, when using the OS/390 Web server.
- Some of the WEB API commands may give a slightly different response when using the Web server Plugin. For instance, when using the command EXEC CICS EXTRACT TCPIP:
  - The TCPIPSERVICE name is returned as NONE.
  - The CLIENTNAME and SERVERNAME are returned only as the dotted decimal forms of the relevant TCP/IP address; therefore, these are identical to CLIENTADDR and SERVRADDR.
  - The SERVERNAME, SERVERADDR, SERVERADDRNU, PORTNUMBER, PORTNUMNU are those of the Web Server, not of the CICS region.

For further information on using and configuring CWS with the CICS WebServer Plugin, refer to the following manuals:

- *CICS Internet Guide*, SC34-5445

For information on configuring the OS/390 Web server, refer to:

1.2.1.3 3270 Web bridge

The 3270 bridge feature of CICS, when used in conjunction with CWS, can provide access to 3270 transactions from the Web. We will refer to this function as the **3270 Web bridge**. To implement this solution, you need only to configure CWS, add CICS PROGRAM and TRANSACTION definitions, and reassemble your BMS mapsets. Most 3270 transactions will then run unchanged, though some applications may require modification. These restrictions are documented in the section “User transaction programming considerations” in Chapter 4 of the *CICS Internet Guide*, SC34-5445. The ease of implementation makes the 3270 Web bridge the preferred solution whenever Web access is required quickly, programming resources are limited, or the application has limited use or life expectancy.

The 3270 Web bridge can be used with either the direct connection to CICS or with the CICS WebServer Plugin. Figure 5 illustrates the data flow for a Web browser request using the facilities of the 3270 Web bridge and a CWS direct connection to access a CICS 3270 transaction. The dark shading indicates the components of the 3270 Web bridge. Note that the 3270 bridge feature is only available when using CICS TS for OS/390 V1.2 or V1.3 or CICS TS for VSE V1.1.

---

**Figure 5. CICS Web support — 3270 Web bridge**
The initial data flow is the same as that described in Figure 3 on page 8 for the description of CICS Web support and the BLI. However, instead of invoking the user program, the Web terminal translation program, DFHWBTTA, is invoked by the BLI. DFHWBTTA starts the transaction to be run in the 3270 bridge environment, where it runs in conjunction with the CICS provided Web bridge exit DFHWBLT. A summary of the components of the 3270 Web bridge follows.

**DFHWBTTA**
This is the Web terminal translation program; it initiates execution of the transaction under the 3270 bridge feature of CICS. DFHWBTTA receives the input in a COMMAREA and formats it to the form in which the 3270 transaction named in the Web user’s input will expect it, attaches the transaction for execution under the 3270 bridge, and waits for it to complete.

**DFHWBLT**
This is the Web bridge exit and is used to control execution of the target transaction. When the 3270 transaction issues a 3270 RECEIVE, DFHWBLT supplies the input from DFHWBTTA. When the transaction running under the bridge sends output to the 3270 terminal, DFHWBLT notifies DFHWBTTA, which translates the 3270 output from the transaction to the HTML equivalent and then returns to the alias program. The alias now resumes standard CWS processing and invokes DFHCCNV for conversion to ASCII and the proper code page, and returns the response to the Web browser.

Note that there are several other sample bridge exits apart from DFHWBLT. These allow invocation from other environments, including MQseries, temporary storage or transient data queues, or a CICS Business Transaction Services (CBTS) environment. Refer to the redbook: *CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge*, SG24-5480, for further details.

**State management**
The program DFHWBST controls the state information required to manage 3270 pseudo-conversations when using the 3270 Web bridge. This information is used by DFHWBTTA and DFHWBLT, the template manager.

**Garbage collection**
The program DFHWBGB is responsible for “garbage collection”. It runs at an interval controlled by the SIT parameter WEBDELAY and purges state data and timed-out bridge facilities associated with terminated 3270 Web bridge transactions.
1.2.2 CICS Transaction Gateway

The CICS Transaction Gateway (CTG) is a set of client and server software components that allow a Java program to invoke services in a connected CICS region. The Java program can be an applet, a servlet, or a stand-alone Java application. We shall describe the architecture of applets and servlets separated as they have distinct characteristics.

The CICS Transaction Gateway is available for production use on OS/390, and on the following distributed platforms: AIX, Sun Solaris, Windows NT, and Windows 2000. It is also available for development use on Windows 95 and Windows 98. It is supported for use with CICS/ESA V4.1 and CICS TS V1.1 (if using the CTG on a distributed platform), and CICS TS V1.2 and V1.3 with any CTG platform. An overview of the CTG function is shown in Figure 6.

For product information on using the CTG, refer to the CICS Transaction Gateway Administration guides for the platform in question. For information on configuring the CTG, refer to the redbook Revealed! CICS Transaction Gateway with More CICS Clients Unmasked, SG24-5277.

Figure 6. CICS Transaction Gateway overview
The CICS Transaction Gateway consists of the components described in the following sections.

**CTG Java gateway application**

This long-running process (or address space on OS/390) is a Java application that functions as a server to CTG client applications such as Java applets.

**Java class library**

This contains the following components:

- **Basic Java methods**
  
  These are used to set up connectivity to a CTG Java gateway application or to invoke the underlying CICS client-server communication mechanism.

- **ECI Java methods**
  
  These methods provide access to CICS COMMAREA based programs in a fashion similar to the native CICS Universal Client ECI.

- **EPI Java methods**
  
  These methods provide a Java API to start CICS 3270 based transactions, and send and receive information from these transactions. They are supported only when using the CICS Universal Client, and thus not when using the CTG on OS/390. There are three distinct EPI interfaces available for use:

  - **EPIRequest object**
    
    This object provides a low level EPI programming interface allowing you to add and delete EPI terminals and send and receive data on these terminals.

  - **EPI support object**
    
    These objects provide higher level methods and are the recommended way of using the EPI. You can control EPI terminals using the provided Terminal, Screen and Field objects. You can also use a provided utility to generate map classes from BMS maps allowing easy access to fields within BMS maps.

  - **EPI Beans**
    
    These beans support development of applications from a number of visual development environments such as Visual Age for Java.
• ESI Java methods

These methods provide a Java API to invoke Password Expiration Management (PEM) functions on a CICS server. The ESI enables client applications to verify or change passwords. This allows revoked userids to be reset from the client application. It is supported only when using the CICS Universal Client and CICS TS or CICS/ESA V4.1.

Transport mechanism
When using the CTG on OS/390, the CTG methods use the function of the CICS EXCI to communicate with the target CICS region. When using the CTG on a distributed platform, the CICS Universal Client is used as the underlying transport mechanism. On both OS/390 and distributed platforms, the CTG interfaces with the underlying transport mechanism using the Java Native Interface (JNI) to invoke the CICS Client or EXCI. This function is provided by the CTGJNI shared library.

Common Connector Framework
Aside from developing your Java application using the CTG provided Java methods, you can also use the IBM Common Connector Framework (CCF) Java Beans. These CCF Beans utilize the functionality of the CTG, which are termed a connector. The CCF model provides the following advantages:

• A common client programming model for connectors. These interfaces can be used together with VisualAge for Java’s Enterprise Access Builder (EAB), to build applets or servlets to access programs or transactions in a CICS region.

• A common infrastructure programming model for connectors, which gives a component environment such as WebSphere a standard view of a connector, and vice versa.

When developing an applet or servlet using the CCF CICS connector, the CICSConnectionSpec, CICSCommunication, and ECIInteractionSpec or EPIInteractionSpec classes are used. These classes can be specified in an EAB Command with an input and output (COMMAREA) to invoke a CICS program.

For more information on developing CTG applications using the CCF, refer to the redbook: VisualAge for Java Enterprise Version 2: Data Access Beans – Servlets – CICS Connector, SG24-5265.
1.2.2.1 CTG — servlet architecture

The CTG servlet architecture enables servlets running within the Java Virtual Machine (JVM) of a servlet engine to use the methods supplied in the CTG Java class library to call a CICS program or to start a CICS 3270 transaction. When using the CTG servlet architecture, the following five components are required.

Web browser
This is a standard Web browser that can send HTTP requests.

Web server
The Web server sends and receives the HTTP requests from the Web browsers. It can be any Web server supported by the servlet engine in use. IBM’s WebSphere Application Server provides a servlet engine that supports a wide range of Web servers including: IBM HTTP Server, Apache Server, Domino Web server, Lotus Domino Go Webserver, Netscape Enterprise server, Microsoft Internet Information Server. On OS/390, the IBM HTTP Server is required by and packaged with WebSphere Application Server for OS/390.

Servlet engine
The servlet engine provides the Java environment on the Web server tier. This is used to run the Java servlet. IBM’s WebSphere Application Server provides a servlet engine compatible with most Web servers.

Servlet
The servlet is written using the CTG Java class library; it is compiled ahead of time and deployed in the servlet engine. The servlet code executes within the servlet engine and sends and receives HTML to the Web browser. The servlet makes ECI, EPI or ESI calls to communicate with the CICS region. It can be invoked by a request from the Web browser using either a URL, an HTML form, or an HTML server-side include.

CTG Java class library
This Java library contains a set of methods, used by the servlet to make ECI or EPI calls to the CICS server. The CTG Java methods are run within the servlet, and requests are sent from the servlet to the CICS region. The class library is supplied in two jar archives, ctgclient.jar (required by all applications that invoke ECI, EPI or ESI methods), and ctgserver.jar (required only by applications that directly invoke the services of the EXCI or CICS Universal Client using the CTG local protocol).
On a distributed platform, the servlet environment can be provided by IBM's WebSphere Application Server in conjunction with any supported Web server. The CTG methods are invoked within the servlet, a connection to a JavaGateway object is created and the servlet then flows the ECI, EPI, or ESI request to CICS using this object.

You can see in Figure 7 that the CTG Java gateway application is not usually required when using servlets on a distributed platform where the CICS Universal Client is also installed. This is because the CTG methods are run within the servlet engine and utilize the CTG local: protocol. The local: protocol uses the JNI to pass the requests onto the underlying CICS Universal Client, which sends the requests to CICS via the LU6.2 connection.

If you do not want to use an LU6.2 connection into your CICS region, you can use the TCP62 support provided by the CICS Universal Client to flow the LU6.2 request over a TCP/IP connection.

Alternatively, if you wish to use a TCP/IP connection to your OS/390 system, you can flow requests from the servlet to a remote CTG Java gateway application on OS/390, using a CTG network protocol from the CTG client running in the servlet to the CTG Java gateway application. This is illustrated in Figure 8.
When using a remote CTG Java gateway application on OS/390, a CTG network protocol is used to flow the requests, from the servlet on the distributed platform, to the CTG Java gateway application running on OS/390. The request is then sent to CICS using the EXCI protocol from the CTG to the CICS region.

This architecture has the advantage that a CTG network connection (most probably TCP or SSL) can be used from the Web server machine to the S/390 system. This can flow over a TCP/IP network, unlike the LU6.2 request, which requires an SNA network. Also, the CTG Java gateway application can exploit the scalability of an S/390 Parallel Sysplex if you wish to handle large numbers or requests. However, the same limitations apply to this scenario as when using a CTG entirely on OS/390. Namely, the servlet is limited to using the ECI methods (so cannot use the EPI or ESI), and must communicate with a CICS TS V1.2 or V1.3 region.

Figure 8. Servlet using a remote CTG on OS/390
Figure 9 illustrates the CTG servlet architecture on OS/390.

On OS/390, the servlet environment is provided by WebSphere Application Server for OS/390. This provides both the OS/390 Web server (IBM HTTP Server) and the servlet engine (WebSphere Application Server). The CTG Java gateway application is not required, since the CTG local protocol is used to pass the requests onto the underlying EXCI.

Since the CTG uses the CICS EXCI protocol, the CICS region and the OS/390 Web server can be situated on different LPARs in a sysplex. However, the servlet is limited to using the subset of the ECI methods that are supported by the EXCI, cannot use the EPI or ESI, and must communicate with a CICS TS V1.2 or V1.3 region.
1.2.2.2 CTG — applet architecture

The CTG applet architecture enables remote Java applets to use the ECI or EPI CTG Java methods to call a CICS program or to start a CICS 3270 transaction. When using the CTG applet architecture, the following four components are required.

Web browser

Applets require a Java enabled Web browser. When an HTML page containing an applet tag is referenced, the applet is downloaded from the Web server. The applet Java methods are then executed in the Web browser Java Virtual Machine (JVM), and create a connection to the CTG Java gateway application using one of the four CTG network protocols, TCP, SSL, HTTP, or HTTPS. These are private protocols and as the names denote, they flow over either a standard TCP/IP socket connection, over an HTTP session, or via SSL encrypted versions of these protocols.

Web server

The Web server serves up the HTML page, which contains the applet tag for the CTG applet, and also serves this applet to the Web browser. There are no special functional requirements for the Web server, other than that it should support HTML and the HTML applet tag.

CTG Java gateway application

This is a long-running Java application that receives the remote ECI or EPI request from the applet and, using the Java Native Interface (JNI), invokes the OS/390 EXCI, or the CICS Universal Client, to pass the request to the CICS region. The Java gateway application is usually on the same machine as the Web server, but can be on a different machine if the Java applet is signed. This is because signing an applet allows an applet to open a network connection to a different machine than that from which it was downloaded.

CTG Java class library

The CTG Java class library provides a set of Java methods that can be used by the Java client (the applet) to open a CTG network connection to the CTG Java gateway application, and then send and receive ECI or EPI requests to the CICS region. The client methods are supplied in the `ctgclient.jar` file which is downloaded from the Web server when the applet is initially downloaded.
Figure 10 illustrates the CTG applet architecture on a distributed platform.

On a distributed platform, the CICS Universal Client is provided as an integral part of the CICS Transaction Gateway. The CICS Universal Client provides ECI, ESI, and EPI interfaces (but note that the ESI protocol cannot be used for requests to distributed CICS systems).

The CICS Universal Client supports LU6.2 connections to OS/390 CICS TS or CICS/ESA V4.1. This requires a supported SNA product to be configured and used. Also supported is the TCP62 protocol for connections to CICS Transaction Server or CICS/ESA V4.1. TCP62 provides support for LU6.2 connections over a TCP/IP network using the protocol encapsulation and translation function of the AnyNet protocol. TCP62 also provides for dynamic configuration of the client SNA node using parameters from the CICS Universal Client initialization file (CTG.INI), meaning no direct configuration is required. TCP62 support is provided by IBM Personal Communications on Windows platforms and on OS/2, or by the CICS Universal Client Host Access feature.

The CICS Universal Client also supports TCP/IP connections to the following distributed CICS servers:

- TXSeries CICS for AIX, Sun Solaris, Windows NT, and HPUX, or OS/2 Transaction Server.
Figure 11 illustrates the CTG applet architecture on OS/390.

On OS/390, the CTG Java gateway application runs as its own address space (or process). It is, however, very similar to the structure on distributed platforms, the principal difference being in the use of the EXCI as the underlying communication mechanism.

The EXCI is invoked by the Java gateway application using the Java Native Interface (JNI), just as the CICS Universal Client is invoked on distributed platforms. The use of the EXCI limits the OS/390 CTG to the subset of the ECI that the EXCI supports, and prevents use of the EPI to invoke 3270 based transactions.

When the CTG for OS/390 V3.1 is used, it is supported with CICS TS V1.2 and V1.3. Note, however, that if you wish to use the CTG V3.1 with CICS TS V1.2, the fix for APAR PQ31270 must be applied to CICS. This does not apply to CICS TS V1.3.
1.2.3 CICS CORBA client support

CORBA is a specification created by the Object Management Group (OMG) that provides a language-neutral standard for interoperability between software objects. Client and server objects communicate indirectly through Object Request Brokers (ORBs) which use a “template” of each object’s interface (methods and parameters) that has been created using CORBA’s Interface Definition Language (IDL).

IDL provides the language-neutrality. Client and server objects on the same machine communicate via a local ORB. Objects distributed across different interconnected machines require an ORB on each machine. The communication sequence is thus: local object to local ORB, to remote ORB, to remote object. The Internet Inter-Orb Protocol (IIOP) provides the message formats and protocols used in this CORBA distributed environment.

CICS TS V1.3 introduces CICS CORBA client support, which allows a client program or object to directly communicate with a Java program in a CICS TS V1.3 region using IIOP. CICS CORBA client support provides the functionality of a CORBA compliant ORB receiving inbound IIOP requests from a client object and invoking the required method on the server object, which in CICS must be a Java program. The CICS Java program can then LINK to other CICS programs or access CICS resources using the JCICS classes. Figure 12 illustrates the architecture of CICS CORBA client support. For further detail refer to the Java Application Development for CICS, SG24-5275-01.

Figure 12. CICS CORBA client support overview
Figure 13 illustrates the flow of requests when using CICS CORBA client support.

1. The CICS TCP/IP socket listener (CSOL) monitors ports reserved for inbound IIOP requests. When a request is received, the listener starts the CICS CIOR transaction; CIOR then calls the CICS IIOP receiver program, DFHIIP.

2. The receiver program, DFHIIP, retrieves the incoming IIOP request and matches its class and method to a template specified in a CICS REQUESTMODEL resource definition. The transaction named in the REQUESTMODEL must specify DFHIOPA as the first program to be invoked. DFHIIP invokes a user replaceable module (DFHXOPUS) to supply a userid and then attaches transaction CIOD.

3. The CIOD transaction is invoked and runs the program DFHIOPA. This issues a CICS LINK to a Java program, DFJIIOP (which can be in a connected region) to handle the request.

4. DFJIIOP analyzes the contents of the IIOP request and then invokes the requested object, supplies the input parameters, and then invokes the requested method on the target object (which must be a HPJ compiled Java program). This Java program can in turn LINK to other CICS programs or access CICS resources using the JCICS classes. Finally DFJIIOP builds the reply and returns it back to DFHIOPA for transmission back to the client.
1.2.4 Host On-Demand

Host On-Demand (HOD) is essentially a full set of terminal emulation facilities that have been written entirely in Java. It allows a Web browser to download terminal emulation applets “on demand” in order to interface with a variety of applications including, 3270 based applications such as those running in CICS. It supports a wide variety of other terminal emulation protocols such as 5250 and vt100.

HOD can offer CICS 3270 terminal users, a quick and easy way to access an existing CICS application environment from a Web browser with absolutely no changes to their existing CICS application environment. Since HOD provides a full function 3270 emulator within a browser there are no restrictions on what type of 3270 or BMS transaction can be supported. Figure 14 illustrates how the main components of HOD interface with CICS.

![Figure 14. Host On-Demand overview](image_url)
Web access to CICS through HOD is principally for intranet or extranet scenarios for the following reasons:

- HOD delivers to the browser a full function terminal emulator, which implies that any user accessing host functions through this method would need to understand the interface of the applications they are accessing, and probably be required to sign-on to CICS using a userid and password. This is unlikely to be something that the general Internet user would be able to do. (The use of the HOD Screen Customizer can simplify this interface for novice users.)

- It will require a Java-enabled browser, often running specific levels of Java, to support the applets you are downloading from the HOD server. This requires a certain degree of control on the types and versions of Web browsers. This does not exclude access through the Internet if you are using the Internet as an external network connection environment for trained or trusted users who can use this path for connecting to your host systems.

The following is a list of the major components in a HOD environment:

- **Host On-Demand Server** — The HOD server provides management of the HOD configuration and supplies the client applets for download. It is installed on the Web server machine and support many platforms from OS/390 to PCs and UNIX hosts.

- **Telnet server** — The telnet server is the key component of the HOD solution. It must be capable of supporting sessions from 3270 clients.

- **Web Server** — Since HOD must be downloaded from a server to clients, a Web Server, such as the IBM HTTP Server, must be installed on the same machine as the HOD server.

- **Java Virtual Machine (JVM)** — Since HOD is a set of Java applets and applications, a Java environment must be established on both server and clients. On Windows NT servers, the JVM is installed along with the HOD Code, but for other types of servers, it must be installed separately. On client machines, the JVM is provided by the browser.

- **Host On-Demand Service Manager** — On Windows NT, this component runs as a service; on other platforms it runs as a Java application and manages the underlying services such as the configuration manager and the redirector on a server, or the locally installed client. The Service Manager must be running for HOD to work.

- **Clients** — Several types of clients are available for download to a terminal from the server. A full description of the clients is given in the redbook *IBM SecureWay Host On-Demand: Enterprise Communications Era Network*.
An abbreviated description of these clients is given below:

- **Download Client** — This is the standard full function client; unlike the cached client, the download client is downloaded each time you wish to use it. This would be used if your browser is unsuitable for the cached client and download time is not a problem.

- **Cached Client** — This is a full function client; which is cached on the local disk the first time you download it. The next time you start a session, the applet does not need to be downloaded; however the server does level-check the downloaded version.

- **Function On-Demand Client** — This is a reduced function version of the Download client and provides only basic function the first time it is downloaded. Additional function is subsequently downloaded when it is required. This client provides faster start-up time.

- **Locally Installed Client** — This is currently only available for Windows 95, 98 or NT, and is installed on the client’s own disk from CD. It would typically be used when a client is connected to a server by a slow telephone line and wishes to avoid the long start-up time.

- **Host On-Demand Express Client** — This is an option on the locally installed client, and works by caching and compressing data to reduce the flow between client and server. It is designed for use on line speeds below 28.8 kbps.

There are a number of other clients that are more suitable for use for debug and test purposes, due to the configuration needed to run them.

- **Screen Customizer** — This is a Java application that enables HOD screens to be extensively modified to provide an enhanced GUI, so that the underlying 3270 screen is not visible to the end user. Screens are captured and turned into maps using the customization studio. These maps can then be displayed by HOD instead of the original screens.

- **Host Access Class Library for Java** — The Host Access Class Library (HACL) for Java provides a set of classes and methods that allow the development of platform-independent applications that can access host information at the data stream level.

- **Host Access Beans for Java** — Host Access Beans for Java provide emulator functions as a set of JavaBeans. They allow developers to rapidly develop custom applications that deliver specific host access functions to their applications.
In addition to using the 3270 emulation sessions, HOD provides the **CICS Gateway** session. This is a specialized emulator session which allows you to access CICS through the facilities of the CICS Transaction Gateway as opposed to using a telnet server. It provides the same 3270 style emulator interface as a standard HOD 3270 client, but in fact, uses the CTG EPI methods within the applet as opposed to the HOD HACL methods to connect to a telnet server. Since the CTG EPI interface is used to build the 3270 screen, the function is limited to the 3270 functionality supported by the EPI.

Complementary to HOD is the IBM SecureWay **Host Publisher**. This is based on the Java HACL function, but unlike HOD, it is a server-side solution, and thus does not require a Java enabled Web browser. It runs on Windows platforms, OS/390, AS/400 and Sun Solaris and is designed for Internet usage. It allows multiple existing host applications (including 3270 based CICS applications) to be integrated and “published” as HTML pages using the Host Publisher Studio. It works in conjunction with WebSphere Application Server and can combine multiple existing screens into single HTML screens. For further detail on Host Publisher refer to the redbook: *Building Integration Objects with IBM Secure Way Host Publisher Version 2.1*, SG24-5385.

When considering how to Web-enable your CICS environment, you should consider solutions that will not only Web-enable your CICS applications, but other application data as well. Using a solution like HOD leaves your options wide open since it can address many applications and environments.

For further details on HOD, refer to the redbook: *IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Networking Computing*, SG24-2149.
Table 1 is intended as a starting point in selecting a solution for Web access to your CICS applications. The first two columns in the table (connection architecture and application interface) reflect the factors that will influence your decision most fundamentally.

First, choose the connection architecture in the first column that applies. In most installations, the network architecture is already in place, and you may want to re-use your investment in a distributed Web server environment, or you may decide to investigate using the OS/390 Web server. Alternatively, you may want to invest in the Web capabilities of CICS itself by using a direct connection to CICS.

Second, in the rows selected from the architecture column, choose the rows in column 2 that apply to applications you want to use from the Web.

Table 1. CICS Web selection table

<table>
<thead>
<tr>
<th>Connection architecture</th>
<th>Application interface</th>
<th>CICS solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Web server</td>
<td>COMMAREA</td>
<td>ECI interface of “CICS Transaction Gateway” on page 14</td>
</tr>
<tr>
<td></td>
<td>3270</td>
<td>EPI interface of “CICS Transaction Gateway” on page 14, or “Host On-Demand” on page 26</td>
</tr>
<tr>
<td></td>
<td>CORBA IDL</td>
<td>“CICS CORBA client support” on page 24</td>
</tr>
<tr>
<td>OS/390 Web server</td>
<td>COMMAREA</td>
<td>ECI interface of “CICS Transaction Gateway” on page 14, or “CICS WebServer Plugin” on page 10</td>
</tr>
<tr>
<td></td>
<td>3270</td>
<td>CICS WebServer Plugin with “3270 Web bridge” on page 12 or “Host On-Demand” on page 26</td>
</tr>
<tr>
<td></td>
<td>CORBA IDL</td>
<td>“CICS CORBA client support” on page 24</td>
</tr>
<tr>
<td>Direct connection to CICS (no separate Web server)</td>
<td>COMMAREA</td>
<td>“CICS Web support — direct connection” on page 8</td>
</tr>
<tr>
<td></td>
<td>3270</td>
<td>CICS Web support direct connection with “3270 Web bridge” on page 12</td>
</tr>
<tr>
<td></td>
<td>CORBA IDL</td>
<td>“CICS CORBA client support” on page 24</td>
</tr>
</tbody>
</table>

A detailed discussion of the CICS Web selection table is available in the CICS Web/selection guide whitepaper, available at:


30 Workload Management for Web Access to CICS
Chapter 2. Workload management technologies

The subject of workload management comprises a range of different technologies and products. However, it can be said that all workload management solutions have the following aims:

- To increase availability so that an unexpected server outage will cause as little noticeable affect as possible. We shall term this **failover**.
- To distribute requests across a group of servers in order to increase the overall capacity and throughput of the system. We shall term this **workload balancing**. There are several different methods of workload balancing, and these can be summarized as follows:
  - One technique is **round-robin**, in which a function that is independent of the users selects a server to handle requests. This approach is better, but it does not take into consideration the current load on the target server or even whether the target server is available.
  - A second approach is to use a simple **advisor** that checks availability (and, perhaps, even the number of connections) on different servers to some extent, before selecting a server.
  - The most sophisticated technique is to have **performance agents** in all application servers that feed managers with statistics; the absence of any statistics also gives an indication of non-availability. The managers, armed with this information, select servers based on the overall service levels that they expect to be delivered.

In order to effectively implement a workload management solution, it will be necessary for certain steps to be taken in your applications:

- Ensure that there is no single point of failure in the end-to-end solution. This applies to both software and hardware components.
- Identify granular elements of a workload that can be distributed among the available computing resources. To achieve this, you will need to consider where “state” is maintained and how this information can be shared across systems. For a detailed discussion on this topic, refer to Chapter 3, “Matters of state” on page 49.

The ability to distribute a CICS workload across available computing resources is particularly important in preventing CPU load becoming a constraint to increased system capacity. This is because of the internal design of CICS. Although CICS TS V1.3 has a multi-domain design, and thus dispatches multiple TCBs, the business logic of a CICS application still all runs within the single QR TCB. Therefore, it is not able to concurrently utilize
multiple processors in a S/390 processor. Thus, to effectively utilize a multi-CPU processor, it is necessary for work to be distributed between several CICS regions.

Conversely, the OS/390 Web server and the OS/390 CTG Java gateway application are both multi-threaded applications. Therefore, they can dispatch work across multiple TCBs, so one address space can utilize several CPUs in a multi-CPU processor. However, since the current S/390 processors can contain up to 12 CPUs per processor, there is still likely to be a point at which running multiple processes of the OS/390 Web server or CTG Java gateway application will provide better throughput than a single process could.

Thus, workload management is also an important issue with these software products as well. In addition, running multiple processes also removes a single point of failure from the system, that you would otherwise have if only using one address space. For further details on TCB usage within CICS, refer to the IBM Redbook, *A Performance Study of Web Access to CICS*, SG24-5748.

Note that workload management is not the same as performance tuning; in fact, the two can be mutually exclusive. Since the very act of distributing work across multiple components is likely to increase end-to-end response time, due to the increased instruction path length required to distribute requests, and the possible extra network latency involved.

### 2.1 MVS Workload Manager

The MVS Workload Manager (WLM) is a tool which is provided to balance the available system resources to meet the demands of S/390 subsystems, such as batch, CICS, TSO, UNIX System Services, and the Web server, in response to incoming work requests.

OS/390 tries to achieve the needs of the workloads (response time) by attempting the appropriate distribution of resources without over committing them. Equally important, OS/390 maximizes system throughput to deliver maximum benefit from the installed hardware and software platforms.

Usually a Service Level Agreement (SLA) is drawn up between the users of a computer installation, and the service manager responsible for running that user’s workload. The SLA will contain agreed-upon values for the following:

- Average transaction response time
- Distribution of response time
- System availability
Workload management can be implemented in one of two ways: compatibility mode or goal mode.

- If compatibility mode is chosen, the response times defined in the SLA are translated into the technical terms of the System Resource Manager (SRM) languages. This requires a high level of skill and detailed knowledge of SRM. Note that IBM has announced that the next release of OS/390 after V2.10 will be the last release to support compatibility mode.

- If goal mode is chosen, the performance criteria is defined in service definitions. There is usually one service definition per sysplex. The service definition itself contains one or more service policies, only one of which will be active at any point in the sysplex. The service definition is a named collection of performance goals and capacity bounds. It is composed of workloads which consist of management classes and resource groups. The skills required to set these up are somewhat easier to attain than those required for compatibility mode.

For more information on this subject, see the IBM Redbook, *OS/390 Workload Manager Implementation and Exploitation*, SG24-5326, and *MVS Planning: Workload Management* GC28-176.

### 2.2 Communications Server

The IBM Communications Server for OS/390 provides support in OS/390 for SNA and TCP/IP. The TCP/IP support contains interfaces sysplex distributor, DNS connection optimization and TCP/IP port sharing, while the SNA support interfaces with the VTAM generic resource feature. All these additional features, when used in conjunction with the WLM, can provide fault-tolerant, high availability systems. We discuss each of these features in this section.

#### 2.2.1 Port sharing

TCP/IP port sharing provides a simple way of workload balancing HTTP requests across a group of cloned address spaces running in the same OS/390 image. For our purposes these could be cloned CICS regions, CTG Java gateway applications, or Web servers.

To enable port sharing, the address spaces are configured to listen on the same TCP/IP port number, and the SHAREPORT parameter is specified in the TCP/IP profile. As incoming client connections arrive for this port, TCP/IP will distribute them across the address spaces that are listening on the shared port. TCP/IP will select the address space with the least number of connections (both active and in the backlog) at the time that the incoming client connection request is received.
This allows you to do workload balancing for incoming HTTP requests across several cloned address spaces. The workload balancing is based entirely on the number of IP connections, and so does not take into account the individual health or capacity of any given CICS region. However, it does provide a very simple means of providing failover and workload balancing across multiple regions within an LPAR. For more information on configuring TCP/IP port sharing, refer to the OS/390 IBM Communications Server: IP Configuration Reference, SC31-8726.

2.2.2 DNS connection optimization

While TCP/IP port sharing allows work to be distributed across several address spaces in the same OS/390 image, DNS connection optimization allows workloads to be distributed across multiple OS/390 images.

DNS connection optimization balances IP connections in an OS/390 sysplex IP domain, by dynamically updating the OS/390 DNS server database based on feedback from MVS WLM about the health of the registered applications. This is sometimes referred to as dynamic DNS, although this feature merely refers to the dynamic update function of the OS/390 DNS server.

When using DNS connection optimization, the client must use the generic host name that is registered with WLM, and then the DNS server will return an IP address based on feedback from MVS WLM. If a new application is registered to WLM, the DNS server will add its IP address to the list of addresses it can return. Conversely, if the application is terminated and is de-registered, WLM will inform DNS of the fact, and the DNS server will remove its IP address from the list of addresses it will return.

Note that all such load-balanced addresses should have a time to live (TTL) value set to zero, which means that every time the name server answers a query from a client workstation, it will instruct the workstation not to cache the answer. However, because DNS connection optimization relies entirely on the DNS server to provide workload balancing, it is limited if the P clients and other DNS servers do not honor the (TTL) value of zero.

DNS connection optimization requires that the OS/390 DNS should be running on at least one OS/390 LPAR in the sysplex. At least one secondary DNS should be configured on another OS/390 LPAR for failover considerations. Without a secondary DNS in the environment, you are exposed to a failure if the OS/390 DNS is not functioning.
Figure 15 shows a diagram of how DNS connection optimization and TCP/IP port sharing can be used together with MVS WLM.

For details on how we configured DNS connection optimization with CWS, refer to 8.1.1, “Configuring TCP/IP” on page 113. You can find out more about connection optimization in a sysplex domain in the manual OS/390 SecureWay Communications Server: IP Planning and Migration Guide, SC31-8512.

DNS connection optimization has the advantage that it is included with OS/390 V2.5 and just requires the usage of a new hostname by all clients. However, its workload balancing function can be easily overridden if the clients use an IP address rather than the generic hostname. This is a common problem, since many clients cache resolved hostnames, and may often ignore TTL values of 0. It also involves an extra round trip for every socket open, since the DNS database has to be queried for every request.

As an alternative to DNS connection optimization, you should consider the function of Sysplex Distributor available in OS/390 V2.10 (see 2.2.4, “Sysplex Distributor” on page 37) or the function of Network Dispatcher and the 2216 router (see 2.6, “Network Dispatcher” on page 45).
2.2.3 VIPA

IBM OS/390 Communications Server addresses the requirement for non-disruptive rerouting around a failing network adapter by allowing the customer to define a virtual adapter with an associated virtual IP address (VIPA). A virtual adapter (interface) has no real existence, and a VIPA is really associated with the TCP/IP stack as a whole.

To the routers attached to the stack via physical adapters, a VIPA appears to be on a subnet on the other side of the OS/390 IP stack, and the TCP stack looks like another router. On the OS/390 IP stack, on the other hand, the VIPA acts somewhat like a loopback address: incoming packets addressed to the VIPA are routed up the stack for handling by TCP or UDP as with any other home IP interface. Dynamic routing protocols can now provide transparent rerouting around the failure of an adapter on the endpoint stack, in that the VIPA still appears reachable to the routing network via one of the other adapters.

Communication Sever for OS/390 V2.8 provides improvements to the VIPA takeover function. VIPA takeover builds on the VIPA concept, but automates the movement of the VIPA to an appropriate surviving stack. There are two forms of VIPA takeover:

- **Automatic VIPA takeover** allows a VIPA address to move automatically to a stack where an existing suitable application instance already resides, allowing that instance to serve clients formerly going to the failed stack/node.

- **Dynamic VIPA for an application server** allows VIPAs to be defined and activated by individual applications (with or without modifying the applications), so that the VIPA moves when the application is moved. This means that application instances on a failing cluster node may be distributed among many (or all) of the surviving nodes, reducing the amount of spare processing capacity that must be provided for each node that may participate in takeover.
2.2.4 Sysplex Distributor

Sysplex Distributor is implemented in OS/390 V2.10 and offers major enhancements to TCP/IP workload management in a sysplex. It has four main functions, which we explain in the following sections.

Balancing of IP packets across multiple OS/390 IP stacks

Balancing is enabled by using a single cluster IP address which routes packets onto multiple nodes. The applications (such as CICS regions or Web servers) must all be listening on the same port on the target nodes. This is similar to the way that Network Dispatcher functions (see 2.6, “Network Dispatcher” on page 45), since the inbound packet is intercepted and re-routed but not manipulated, and the outbound packet is not modified at all.

Close integration with MVS WLM

Sysplex Distributor provides for close integration with the MVS WLM policy agent and Service Level Agreements (SLAs) in making the routing decisions. This is different from Network Dispatcher, since it has to poll the WLM advisor on OS/390 to update its routing information.

Failover of the cluster IP address

The cluster IP address is actually a VIPA (virtual IP address) and so can be dynamically routed to another OS/390 LPAR in the sysplex. This allows for failover of the cluster address. Note that VIPA takeback also allows for dynamic takeback of the address to the original adapter; this is similar to the hot standby function of Network Dispatcher.

Re-routing of IP packets away from one node

The cluster address can be instructed to no longer route further packets to a certain virtual IP address. This allows work to be directed away from a CICS region or Web server which is to be quiesced and onto another region on a different IP stack. This allows the LPAR to be quiesced, without disrupting any of the inbound IP packets.

Non-disruptive movement of server applications can also occur when used in conjunction with Sysplex Distributor. By utilizing the non-disruptive VIPA capabilities, the impact of planned outages can be greatly reduced. Previously, OS/390 V2.8 did not provide for an immediate takeback of the VIPA. The Backup VIPA in the OS/390 V2.8 TCP/IP stacks would retain control of the VIPA until all sessions end. At that point, the VIPA would move to the back to the owning TCP/IP stack.
What this means to workload management is that there is now a workload distribution and failover mechanism that is positioned at the entry to the sysplex. Unlike DNS connection optimization, this allows dynamic workload balancing without the overhead a DNS query, and allows for packets returning from the selected server to efficiently routed back through the distributing stack. For more information on both Sysplex Distributor and dynamic VIPA, refer to the latest version of the IBM Redbook *TCP/IP in a Sysplex*, SG24-5235.

### 2.2.5 VTAM generic resource

The VTAM generic resource function is the key to effective workload balancing in an SNA LU2 environment. The generic resource function allows the assignment of a generic resource name to a group of CICS regions that all provide the same function. The generic resource name is assigned to multiple active CICS regions simultaneously, and VTAM can automatically distribute sessions among these regions rather than assigning all sessions to a single region.

Thus LU2 session workloads in the network can be balanced dynamically across a group of CICS regions. Session distribution is transparent to the end user; an LU initiates a logon request using the generic resource name and need not be aware of which particular CICS region the session is actually bound to. In this way, the VTAM generic resource function can also increase application availability, since each active CICS region that uses a given generic resource name can back up other generic resource members. When a generic resource member fails, an LU can reinitiate its session using the same generic resource name. VTAM resolves the session initiation to one of the other generic resource members.

However, the VTAM generic resource function is not as useful in an LU6.2 parallel session environment (as used by the distributed CTG), since it operates at the Physical Unit (PU) level, so all sessions from the CICS Universal Client machines will be bound to the same CICS region. This provides failover in case a CICS region fails, but does not provide a mechanism for balancing ECI or requests from the CTG across a group of CICS regions.

For more information on the VTAM generic resource function, see the *OS/390 IBM Communications Server: SNA Network Implementation Guide*, SC31-8563.
2.3 CICS Transaction Server

CICS Transaction Server for OS/390 (CICS TS) V1.3, contains the richest set of functions compared to any other previous release. Consequently, CICS workloads can now originate from many more sources than before. Additionally, although the traditional hub routing model still exists, where work originates from one region (usually the terminal owning region, or TOR) and is then distributed to run in one or more application owning regions (or AORs), CICS TS V1.3 introduces a new routing model where work can originate in any region, and also be routed to run in any region. This is known as the distributed routing model.

Workload management in CICS TS is achieved through the process of dynamically routing work between several CICS regions. Since you will have to operate and maintain several CICS address spaces for this purpose, you can choose either to run them as separately connected systems in an MRO complex, or you could define them as CICSPlex System Manager (CICSPlex SM) managed regions.

CICSPlex SM has many advantages, the most obvious being the provision of a single point of control for all your CICS regions. Additionally, while CICS provides its own user-replaceable routing program, DFHDYP, which you would have to customize to balance your workload dynamically if using MRO, the workload balancing capabilities of this program are very limited. In contrast, CICSPlex SM provides its own routing program, EYU9XLOP, to dynamically route your workload, and it is an easier process to implement. For more information about the new features of CICS TS 1.3, see the IBM Redbook, *CICS Transaction Server for OS/390: Version 1 Release 3 Implementation Guide*, SG24-5274.

2.3.1 CICSPlex SM

CICSPlex SM is an integral product supplied with CICS TS 1.3 which provides several tools to facilitate the management and operation of a CICSPlex. The tools provided include:

- **Workload Manager (WLM)**, which provides the implementation of dynamic routing for workload balancing and workload separation. In a CICSPlex, workload balancing is defined as dynamic routing of requests across all available regions, whereas workload separation is a type of round-robin distribution where criteria such as terminal identifier, or transaction name are used to route the request to a region (or set of regions).
• **Business Application Services (BAS)**, which provides a more controlled approach to CICS resource management and can be used either in conjunction with or to wholly replace the CEDA function.

• **Web User Interface (WUI)**, which enables access to a subset of CICSpLex SM functions from a Web browser. For an example of a CPSM WUI screen, see Figure 82 on page 133.

Clearly, in a workload managed environment, the WLM is the most significant of these tools, and the workload models which it controls have introduced new terminology to the CICS arena.

Traditionally, a CICSpLex environment has consisted of one or more terminal-owning regions (**TORs**) connected to a group of application-owning regions (**AORs**). This is known as a hub model, and the design assumes that the workload originates in the TOR and is then routed to run in one of the AORs. Thus, in a hub model, only the TOR is a router and only the AORs are targets. Since the advent of CICS Web-enablement, the term TOR is no longer always appropriate, since work is not initiated from a traditional 3270 CICS terminal but from Web browser, which can be connected to CICS in a variety of ways. Requests can arrive either via the CICS TCP/IP listener, via an EXCI connection, or via a traditional LU6.2 connection. In this book we will term any region participating in receiving such Web requests as a listener region.

Since CICS workload input is no longer limited to the TOR, there is also now an alternative to the hub model, the distributed model. In the distributed model, any region can be considered to be both a router and a target region, and can route to any other target region. From a CICS Web point of view, there are some restrictions to this: EXCI requests, for example, must always come from a predefined connection, and HTTP requests must always come from a predefined TCPIPSERVICE, however, the called program can then be dynamically routed anywhere else in the CICSpLex.

There are two types of performance algorithms which the CICSpLex SM Workload manager supports: the **queue** algorithm and the **goal** algorithm. The aim of the queue algorithm is to select the target region that:

- Is the healthiest
- Has the least queue depth (or load)
- Has the fastest CICS link from the router region
- Has the lowest calculated transaction abend probability
The aim of the goal algorithm is to select the target region that:

- Is the healthiest
- Has the least load
- Has the fastest CICS link from the router region
- Has the lowest calculated transaction abend probability
- Is the most likely to allow the transaction to meet the response time target goal set for it and other transactions in its workload management class

2.3.2 Dynamic routing

CICS dynamic routing was originally only available to terminal initiated workloads, and so was termed dynamic transaction routing. However, the dynamic routing program in CICS TS V1.3 now supports dynamic routing for additional workload elements, including the CICS-to-CICS dynamic program links (DPLs). This means that DPL requests can now be workload balanced within your CICSPlex, rather than having to be routed to the same AOR, as was previously the case.

For a distributed program link (DPL) to be eligible for dynamic routing, the remote program must either be defined to the local system as DYNAMIC(YES) or not be defined to the local system. If the program is not defined in the local system and program autoinstall is inactive, the dynamic routing program is invoked. If the program is not defined in the local system and program autoinstall is active, however, the autoinstall user program is invoked. The dynamic routing program is then invoked only if the autoinstall user program installs a program definition that specifies DYNAMIC(YES) or does not install a program definition.

2.3.3 VSAM RLS

File sharing is an important part of any CICS workload management solution, since just about all CICS transactions involve file access, either to VSAM files or perhaps IMS, DB2, or other databases.

VSAM Record Level Sharing (RLS) is a function made available in DFSMS V1.3, and exploited by CICS TS. It enables VSAM data to be shared, with full update capability, between multiple applications running in multiple CICS regions in a Parallel Sysplex. This removes the affinity to VSAM data which had previously existed in earlier releases of CICS, and removes the necessity for using a single file-owning region (FOR) to access all VSAM files. For more information on RLS, see DFSMS/MVS V1R5 Access Method Services for the Integrated Catalog Facility, SC26-4905.
In previous releases of CICS, it was common to utilize a file-owning region (FOR) or queue-owning region (QOR) in order to implement file sharing. Although this is still a supported configuration, it introduces a single point of failure in the FOR or QOR. Instead, the capabilities of a coupling facility can be utilized. VSAM RLS can now be used to share files, and a temporary storage data sharing server to share temporary storage queues.

2.4 CICS Transaction Gateway

The CICS Transaction Gateway (CTG) does not itself provide any workload management functions. However, it does utilize the underlying technologies of the External CICS Interface and the CICS Universal Client, both of which have workload management capabilities.

2.4.1 CICS Universal Client Workload Manager

The CICS Universal Client workload manager is a utility provided on Windows NT that allows ECI or EPI requests to be balanced across a group of CICS regions. Options are provided to allow you to balance work across CICS regions using either a round-robin technique, or a weighted distribution. It also provides the ability to detect failed regions, and provides a configurable time-out period to check the status of regions that have previously failed. It does not, however, provide any form of performance agent to feedback about the status of the CICS regions. Thus, it is best viewed as a means of removing a single point failure in a listener region.

Note that the CICS Universal Client Workload Manager is not available with the CICS Universal Client on AIX, Solaris, and OS/2, but the exit on which it is based is provided, enabling you to implement your own customized Workload Manager. For further details, refer to CICS Universal Client for Windows Administration, SC34-5449.

2.4.2 EXCI

The CTG on OS/390 uses the facilities of the external CICS interface (EXCI) to route ECI requests from the client (servlet or applet) to the target CICS region. The EXCI is an application programming interface that enables an OS/390 program running in a non-CICS environment (a client program) to call a CICS program, and to pass and receive data by means of a COMMAREA. The CICS application program is invoked as if linked-to by another CICS application program. The client application allocate and open sessions (termed pipes) to a given CICS region, and pass requests over these pipes. The multi-region operation (MRO) facility of the CICS inter region
communication (IRC) facility supports these requests, and each pipe maps onto one MRO session, with a limit of 100 pipes per EXCI address space.

DFHXCURM is a user-replaceable module supplied by CICS, and it allows the destination CICS APPLID on and EXCI call to be altered and various retryable errors to be handled. This allows for basic workload balancing of EXCI calls, based on a simple availability check to be performed in this exit before the EXCI call is sent to the CICS system. For further information on DFHXCURM, refer to 5.1.3, “EXCI” on page 82. For more information about the EXCI in general, see the CICS External Interfaces Guide, SC33-1944.

2.5 WebSphere Application Server

In this section we discuss the workload management options available in WebSphere Application Server (WebSphere AS) both on OS/390 and distributed platforms.

2.5.1 WebSphere on distributed platforms

WebSphere AS is supported on the UNIX, Windows, and AS/400 platforms. The main components of WebSphere AS are the Web server (HTTP server), the Web server plug-in, the servlet engine (which contains the JVM), and the administrative server. Using these components, WebSphere provides a rich set of techniques for workload balancing, as follows:

Cloning servlet engines

WebSphere cloning provides a mechanism for scaling of WebSphere Application servers, whereby clones of servlet engines can be created. These provide identical, yet independent JVMs in which the Java applications run. Using this technique, multiple clones can be created on the same physical machine in order to fully utilize the processing power of a multi-CPU machine.

OSE remote

OSE is a communication protocol private to WebSphere. It is used for communication between the Web server plug-in and the servlet engine. If the Web server and the servlet engine are running on different machines, the OSE remote protocol can be configured to run over a TCP/IP connection between the machines. This technique can be combined with cloning so that requests from the Web server can be distributed across clones located on different machines.
Servlet redirector

The servlet redirector is a mechanism that allows HTTP request received by the Web server to be forwarded to one of several application servers. The forwarding mechanism uses the EJB facility of WebSphere and forwards requests using the RMI/IIOP protocol.

EJB WLM

EJBs deployed in WebSphere AS can take advantage of the WebSphere Workload Management facility for EJBs. This utilizes a smart stub on the client that automatically and transparently forwards the request to any of a set of available servers. The smart stub communicates with the WebSphere Enterprise Java Server runtime to keep track of which servers and EJB instances are available at any given time.

For more information on the workload management options when using WebSphere AS on distributed platforms, refer to the IBM Redbook, *WebSphere Scalability: WLM and Clustering Using WebSphere Application Server Advanced Edition*, SG24-6153.

2.5.2 WebSphere on OS/390

On OS/390 WebSphere AS is supplied together with the OS/390 Web server (the IBM HTTP Server). The OS/390 Web server can be configured to run in two modes, either normal mode or scalable mode.

In scalable mode, OS/390 WLM is used to balance work across a group of Web server address spaces, namely, the queue manager and one or more HTTP Q servers. This offers a number of advantages over a single address space design, including these:

- **Scalability** — A single address space is limited in the number of tasks that it can run and the amount of virtual storage available to it. Allowing work to be spread over multiple address spaces allows significantly more work to be accepted and processed.

- **Availability** — You can set up your server to separate different types of work into different queue server address spaces, such as GWAPI programs, CGI scripts, simple HTTP GETs, and servlets that invoke CICS. Then, failures in certain programs will not result in an outage of your Web server.

For further details refer to *OS/390 e-business Infrastructure: IBM HTTP Server 5.1- Customization and Usage*, SG24-5603.
2.6 Network Dispatcher

Network Dispatcher is an IBM solution that provides an advanced IP level workload balancing mechanism. It is supported on a variety of workstations including Windows NT, AIX, Solaris and Linux, and is provided as part of the WebSphere Edge Server. For detailed information refer to the following publications:

- The IBM Redbook: *IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher*, GC31-8496
- The IBM Redbook: *TCP/IP in a Sysplex*, SG24-5235.

We will now provide a brief summary of the following three major components of Network Dispatcher:

- Dispatcher
- Content Based Routing
- Interactive Session Support

2.6.1 Dispatcher

The Dispatcher is the key component of the Network Dispatcher solution. It can provide workload balancing for any TCP or UDP protocol, including HTTP requests to a Web server or TCP packets to an application such as the CICS Transaction Gateway. The workload balancer node functions as the cluster IP address for all your target IP servers. You then define the ports you want to support inside each cluster and then the actual server that will provide the service on each of those ports.

Each of the servers to which the Dispatcher load-balances must be one hop away from the Dispatcher cluster (that is with no intermediate router), and also must have the IP address of the cluster configured as an alias for its own loopback adapter. Without this alias, packets would be rejected by the server, as the destination address in the IP packet would not match the server's IP address. Once the server has processed the inbound IP packet, it establishes a connection back to the client by standard TCP/IP semantics, by simply swapping the source and target addresses as supplied in the TCP/IP packet.
This means that the server replies to the client with the cluster IP address.
As a direct result, the balancing function is invisible both to the client and the
clustered server, meaning that Dispatcher is not dependent upon server
platforms, or function. This also has the advantage that outbound traffic
(which is often much larger than inbound packets) does not need to flow
through the Dispatcher machine and in fact can even take a different physical
route via a high bandwidth connection.

The Dispatcher decides on the best target IP address for each packet, based
on feedback from the servers. The Dispatcher has a Manager function which
periodically sets the weights used for routing packets. When you configure
Dispatcher, you assign the proportions of the following four metrics:

- Count of active TCP/IP connections for each server on each port.
- Count of new connections for each server on each port since the last
  interval started.
- Feedback from Advisors. Standard Advisors (for protocols such as HTTP
  or Telnet) are shipped with Dispatcher and provide an application-level
  check that each server is running. Custom Advisors (such as customer
  CICS program) could also be used to provide more specific and tailored
  load metrics directly from the application.
- Feedback on available capacity from the WLM advisor (OS/390 Workload
  Manager), Dispatcher Observer mode of ISS (on AIX, Solaris, and
  Windows), or by the System Monitoring Agent (Linux).

Server affinity
This function is sometimes referred to as the "sticky option". Server affinity
allows workload balancing for those applications that preserve some kind of
persistent state across separate connections, without losing this state when
the client reconnects. This causes all subsequent requests from a particular
client to be resent to the same target server. Although this can avoid the
problem of having to share state information across servers, it can cause
exceptionally high loads on particular servers, since HTTP proxy servers will
cause the source IP address to be modified to that of the HTTP proxy itself.

High availability
Network Dispatcher provides fault tolerance for the workload balancer by
allowing for a secondary or standby machine. This machine can either share
the load (if using a different cluster address) or act as a standby in case of
failure.
2216 router
The IBM 2216 router also implements IP level workload balancing and incorporates an internal version of Network Dispatcher that interacts with the OS/390 Workload Manager advisor. However, the IBM 2216 router has been withdrawn from marketing following IBM’s alliance with Cisco, and equivalent function is now implemented in version 3 of Network Dispatcher. For further information on the Network Dispatcher function in the IBM 2216, refer to the IBM Redbook *TCP/IP in a Sysplex*, SG24-5235.

2.6.2 Content Based Routing
The Content Based Routing (CBR) function of Network Dispatcher combines the workload balancing function of Dispatcher with the IBM Web Traffic Express caching proxy server to permit workload balancing based on the content of HTTP requests. In effect, this means that the server destination can be chosen by analyzing the information in the HTTP data stream itself.

2.6.3 Interactive Session Support
Interactive Session Support (ISS) is a Domain Name Server (DNS) based component of Network Dispatcher. It provides a load-monitoring daemon to be installed on each of the servers that form part of your installation. Information from these daemons are used by ISS Name Server to allow it to decide which IP address to give a client based on the load on the candidate server.
Chapter 3. Matters of state

In any conversational application, there is always a concept of **state**, that is, information that allows each partner in the conversation to know how to proceed. Just as human memory stores short-term information during a conversation, two interconnected computer devices must also store short-term state to enable the conversation between the two devices to proceed.

**State in CICS pseudo-conversations**

Traditionally, CICS applications were designed as pseudo-conversational chains, that is, with many separate tasks connected via state data, and which appeared to the user as one long conversation. This has the advantage that the CICS transaction is not suspended while the user is thinking, and so less resources are consumed within CICS. Instead, the state of each leg of the conversation is stored and managed within CICS, and it is this state that ties the legs of the conversation together into a meaningful dialog (Figure 16).

**Figure 16. CICS pseudo-conversational processing**

In CICS, the state of such a transactional conversation is usually maintained by the application, which for a 3270 based application would typically use information associated with the 3270 terminal, such as the COMMAREA or
the Terminal Control Table User Area (TCTUA) to control the next leg of the conversation, in conjunction with the next TRANSID (specified using the EXEC CICS RETURN TRANSID command). Non-3270 based transactions may use other techniques such as temporary storage queues or GETMAINed storage areas.

All of these techniques are likely to introduce an affinity between the different tasks that make up the legs of the conversation; this is termed an inter-transaction affinity. Such inter-transaction affinities pose a significant problem for CICS workload management solutions, since the affinity between the two transactions will require that subsequent legs of the pseudo-conversation must run in the same CICS region. To help address this problem, you should refer to the CICS Transaction Affinities Utility Guide, SC33-1777. This guide describes the usage of the CICS Transaction Affinities Utility, which can scan CICS load modules for EXEC CICS commands that are likely to cause affinities.

**State in Web applications**

Similarly, Web based applications must also maintain state, to allow each series of Web pages to be inter-linked into a useful application. This is somewhat harder in the Web environment, since each HTTP request is stateless and (unless persistent HTTP connections are in use), will involve the setup and tear-down of a TCP/IP socket for every HTTP request.

Consider, if you will, an on-line share trading application. In its simplest form, the user must identify themselves, select the desired company in which to trade, receive information on the current price, and perhaps other holdings, and then submit their order. At each stage the application needs to know the information from the previous stage, and it needs to be able to prevent this information from being intermixed with any other transactions that are being processed at the same time. Since the Web is based on the HTTP protocol, which is inherently stateless, the application must use its own mechanisms for passing this state information from one leg of the conversation to the other.

Usually the presentation logic of a Web application would run within the Web server (or connected CICS system), and be driven by a CGI script, or perhaps a Java servlet or JSP. At the end of each leg of the conversation, the presentation logic would need to pass to the Web browser all the state information about the conversation so far (customer name, company chosen, and so on), and the Web browser would need to flow this information back to the server with each subsequent request.
This then poses the problem that potentially large amounts of information would be required to be flowed across the network, when it would much better be left on the Web server. So instead of flowing such information with each and every request, it is possible to store the state data on the Web server and flow only a unique identifier (state token) with each request. This state token is used by the server to identify any given request, and to retrieve the correct state data from storage on the Web server, and so respond in the correct fashion. This is illustrated in Figure 17.

Figure 17. Web server state — showing state tokens

If you use CICS Web support for the Web-enablement of your CICS application, then your CICS Web application logic is also likely to require that state data is stored within CICS. For further discussion on this subject, refer to 3.2, “State in CICS Web-enabled applications” on page 55.
3.1 State management techniques

Moving on, what practical techniques are available to address the storage of state data in a Web applications? There are three main techniques in use today on the Internet: URL rewriting, HTTP cookies, and HTML hidden form fields. We will look at each of these in turn, and reflect on how each can be used with each of our CICS Web-enabling technologies.

3.1.1 URL rewriting

This technique was (and still is) widely used in Web CGI based applications. It preserves all the state data by storing it within the URL of each HTTP request. Thus the information is clearly visible to the end user, but is also readily accessible to the Web application. An example of such a rewritten URL is shown in Figure 18. Each element of data is specified in the query string of the URL. The parameter list begins with a question mark (?) and consists of a set of name=value pairs separated by an ampersand (&).

```
```

Figure 18. HTML URL rewriting

In our example you can easily identify the user (Cavin), the company selected (Casey Import Export), and the requested action (getQuote=submit) and the host machine (nassau). When the Web server program receives this information it would be able to immediately identify the user, and the company selected, and decide which screen to display next.

URL rewriting is still widely used in Web applications, but can involve the transmission of significant extra information in the HTTP datastream, which can easily be viewed or modified by the user, and is at the very least unsightly. It is also limited in that only a relatively small amount of additional data can be held in the URL query string.

From a workload management viewpoint, URL rewriting in its simplest form does have the advantage that it allows all the state information to be stored in the URL which is passed back and forth between the Web browser and server. Thus, it allows requests to be workload balanced across different Web servers (or CICS regions) without concern for the location of the stored state information, which is always located within the URL. However, it is also quite possible to combine this technique with the usage of state tokens and just flow a token from the browser to the Web server.
3.1.2 HTTP cookies

An HTTP cookie is a small text-based document containing fields of data on separate lines. This is created by the Web server and sent to the browser as an additional part of the HTTP header. Most browsers allow cookies to be enabled or disabled, or their usage customized. The Netscape browser caches session-based cookies in memory, and stores persistent cookies (those that persist across browser sessions) in the file cookies.txt.

An example cookie is shown in Figure 19. A cookie consists of the following information, although the name=value pair is the only required attribute:

name=value

Both name and value can be any character strings, and can be up to 4 KB.

expires=date

This sets the date when a cookie becomes invalid. The date is formatted as in the following examples: Thursday, 12-Oct-2000 10:43:00 GMT. If no date is set, then the cookie is a per-session cookie, and will be discarded when the Web browser session ends (that is, when all instances of the browser process terminate).

path=pathname

This specifies a range of URLs for which the cookie is valid. If it is set to /tradercv, then the cookie will be sent from the client to all servers, when the URL is of the form:

http://<hostname>/tradercv/*

domain=domain-name

This attribute specifies a domain name range to which cookies stored on the browser can be sent. The domain name must contain at least two dots, (for instance .ibm.com.) This would specify that the cookie would be sent to servers with the hostname of wtsc61.itso.ibm.com or www.ibm.com but not to www.yahoo.com., for example.

secure

The secure attribute tells the client to only return the cookie over a secure (SSL) connection.

```
User=Cavin&Company=Casey Import_Export&getQuote=submit
path=/tradercv
expires=13-Oct-2000 00:00:00 GMT
domain=itso.ibm.com
```

Figure 19. A sample HTTP cookie
Cookies can be used either to store all the state required information about the user, or can be used merely as state tokens, which are then used as unique identifiers to refer to the relevant information on the server. Cookies can be used from any application that has access to the HTTP data stream. For CICS Web access, this includes all servlets (which can access CICS via the CTG or CICS CORBA client support) and Web-aware CICS Web support applications.

For further information on cookies, refer to http://www.cookiecentral.com or to RFC2109, which can be obtained from the http://www.ietf.org/rfc.html Web site. For a very interesting insight into different possible uses of cookies, http://www.w3.org/Security/FAQ/wwwsf7.html#Q66 provides a good reference.

3.1.3 Hidden HTML form fields

This technique preserves state data by storing it within each HTTP request. The data is embedded into an HTML form by the server and sent to the browser. However, it is not displayed by the browser, but is resent back to the server when the HTML form is resubmitted by the user. The data is hidden using the input type=hidden tag, as shown in Figure 20. Each element of data is specified as an HTML hidden data type.

```
<form name=getQuotes method=post
<input type=hidden name=_username value=Cavin>
<input type=hidden name=_company value=Casey_Import_Export>
```

*Figure 20. Hidden HTML form field*

Using this technique allows large amounts of information to be stored in the actual HTML sent back to a Web browser. This information is then sent back to the subsequent server, which receives the result of the next HTTP POST. This has the advantage that no state need be stored on the server, and therefore requests can be balanced across different servers without consideration for storing state.

However, it is also possible to combine this technique with the usage of state tokens, such as provided for in the use of WebSphere AS sessions. This unique state taken would be generated by the server, and only this would then be passed in the HTML form field. This token would then be used on a subsequent access to retrieve the state information stored on the server. This technique is the design used by our Web-aware CICS Web support Trader sample application, used in Chapter 8, “CICS Web support” on page 111.
3.2 State in CICS Web-enabled applications

Now that we understand the different techniques available for managing state in Web-enabled applications let us now look at how to utilize these techniques with Web-enabled CICS applications.

3.2.1 State in CWS applications

If you are designing a CICS Web support (CWS) application, you will face similar issues when deciding how to handle the state of the Web application across the different transactions in a Web conversation, since the CICS region is responsible for the receiving and handling of the HTTP datastream. If you intend to use the 3270 Web bridge solely, then your task is eased, since the 3270 bridge handles state internally, using a hidden HTML form field called DFH_STATE_TOKEN. However, if you intend to write your own HTTP presentation using the facilities of CWS (we term this Web-aware), then you will need to design a mechanism to handle state within CICS.

Web-aware applications

Your first choice will be what mechanism to choose, in order to store any state data in your CICS region(s). In CICS 3270 applications it is common to store application state data in a variety of region controlled storage areas such as COMMAREAs, GETMAINed storage, temporary storage (TS) queues, the common work area (CWA) or the TCTUA. Often, these would use the terminal identifier as the unique identifier. However, when using a CICS Web-aware application your application does not have a terminal, and therefore it is not possible to use such a mechanism. Instead, you will need to use a technique similar to that used by the CWS sample state management utilities, DFH$WBST and DFH$WBSR. These store a given record in either GETMAIN storage or temporary storage queues, and return a unique state token for its subsequent retrieval.

This state token should be passed from the CICS application to the Web browser using one of the techniques previously discussed for passing state from a Web browser to a Web server (URL re-writing, HTML hidden form fields, or HTTP cookies). To assist you in this task, you may want to use the CWS parser program DFHWBP A. This will parse a string for the value of a given name=value pair. For further details on this utility, refer to Appendix G in the CICS Internet Guide, SC34-5445.

In addition, if you wish to workload balance your Web requests across multiple CICS listener regions, you will need to store the state data in a place that is available to any CICS listener region, and ideally shareable across your sysplex. Shared temporary storage is a good solution, since it is easy to
set up and it provides access to shared storage in the coupling facility. All that is required is a temporary storage data sharing server running on each LPAR, and just one TSMODEL definition in each CICS region (Figure 21). The data is physically stored in the coupling facility and can be accessed from any CICS region in the sysplex. Furthermore, if one CICS region terminates, the data is not lost, and is still available from the other regions.

![Diagram of sysplex with TCP/IP load balancing, CICS Listener Region, TS Data Sharing Server, and TS Pool](image)

**Figure 21. Shared temporary storage, for storing Web state data**

Whatever mechanism you use for state storage, you will also need to consider a means of deleting redundant state records, since it is quite possible (even likely) for a Web transaction to be terminated in full flow, which will result in redundant state records being left behind. This could potentially lead to the TS pool (or other storage area), becoming full and preventing creation of any more new state tokens. For details of the program (DELTSQS), which we developed to browse and discard redundant temporary storage queues, refer to Appendix B, “Using the additional material” on page 221.

Alternatively, it would also be possible to use other shared repositories for the sharing of state data across CICS regions, including Coupling Facility Data Tables, VSAM RLS, DB2 or IMS data sharing, CICS Business Transaction Services, or even a file or queue-owning region. For a full description of how we actually implemented this technique in our CWS scenario, refer to Chapter 8, “CICS Web support” on page 111.
3.2.2 State in CTG applications

When Web-enabling applications using the CICS Transaction Gateway (CTG), your HTTP presentation logic will be written in Java. There are several different types of Java application design, and these all have very different means of handling state. We will consider Java applets, standalone Java applications, and Java servlets.

Java applets

A Java applet is downloaded from a Web server to the Web browser machine and executes in the JVM of the Web browser, thus the HTTP presentation logic all executes on the client machine. Therefore even if the Java applet makes multiple calls to CICS using the CTG Java methods, the state can be handled within the applet itself, since it always knows exactly what calls it has made. It may also be desirable to store some state in the CICS applications, and this can be done so using similar techniques as discussed in 3.2.1, “State in CWS applications” on page 55. However life is generally easy for the application designer when it comes to dealing with state in Java applets.

Java applications

Java applications can have many different designs, but typically an application would use the Java awt or swing classes to build a GUI with which the user interacts. Requests can then either be sent to a remote CTG or to a local CTG, before they are flowed to a CICS region. However, like applets, the state can be stored internally within the Java application and so is not usually an issue of concern.

Java servlets

Java servlets execute within the JVM of a servlet engine provided by a Web server, an example being IBM’s WebSphere Application Server. Since the HTTP presentation logic has to execute on the Web server tier, managing state is a prime concern when developing a servlet solution. Since the servlet has access to the HTTP datastream it can use the traditional means of storing state previously discussed (URL re-writing, hidden HTML form fields, or HTTP cookies). However, the servlet API also provides a unique means of handling state using sessions as provided by the servlet API.

Servlet sessions are an implementation of the javax.servlet.http.HttpSession interface described in the Java Server API specification. The underlying session support they rely on can use either HTTP cookies or URL re-writing, but the interface to the servlet application is the same. We will consider the use of cookies, as they are perhaps the most popular technique. You can find more information on servlet session management in “User sessions” on page...
Each instance of a servlet can create a unique session-ID for each stateful session. The state information is stored as an object in the servlet, and is tied to a unique session-ID (state token) that is sent back to the Web browser as an HTTP cookie. On a subsequent request, this cookie is sent back to the servlet, which can then obtain the state information from the specified object.

However, the use of such a configuration poses a problem if you wish to workload manage requests across several servlets running in different Web servers (such as WebSphere Application Server) using an IP spraying technology such as Network Dispatcher. This is because the state information is only stored within each instance of an Application Server. In Figure 22, which illustrates this concept, you can see an HTTP request originating from client (1), and being routed by Network Dispatcher to a WebSphere AS (1). The request causes session data to be stored in WebSphere AS (1) for future use by the client (1).

Figure 22. Session based state data
When a subsequent HTTP request is made from client (1), Network Dispatcher may route the request to WebSphere AS (2). However, there is no session data for client (1) residing on WebSphere AS (2) and so the request to access the state data will fail.

To avoid this situation, the session data can be stored in a persistent data store. WebSphere provides support for a DB2 database for this purpose. This is illustrated in Figure 23.

![Figure 23. Session based state data 2](image)

When the first HTTP request is made, state data is made persistent by WebSphere AS, in a DB2 table. Upon the second request, Network Dispatcher routes the request to WebSphere AS (2). WebSphere AS (2) will then access the original state data from the DB2 table.

This configuration allows a servlet solution using WebSphere AS to be horizontally scaled across different machines according to the needs of the Web site.
3.2.3 State in CICS CORBA applications

A CICS CORBA solution involves a client ORB communicating with a server ORB within CICS, and probably invoking existing business logic in CICS applications.

The client ORB can run in a wide variety of different environments, but is most likely to be a Java application (or perhaps applet), or a Java servlet. Therefore, the same considerations for management of state apply to using these technologies as when using the CTG. See 3.2.2, “State in CTG applications” on page 57 for further details.

However, since the server ORB runs within a CICS region, you may also need to store some state information within the CICS region to pass data between IIOP calls. If you are workload balancing IIOP calls across different CICS regions, you will need to utilize a technique that allows the sharing of data between CICS regions. Techniques such as shared temporary-storage queues should be considered to facilitate this. For further details, refer to 3.2.1, “State in CWS applications” on page 55.

3.2.4 State with HOD

A HOD solution involves the Web-enabling of existing 3270 based applications, and as far as the CICS 3270 application is concerned the 3270 terminal that HOD uses is a real 3270 terminal. Therefore, any state management considerations based on the 3270 terminal will remain unchanged if the application is Web-enabled using HOD.
### 3.3 Summary

Table 2 compares each of the state storage techniques and contrasts the advantages and disadvantages of each, with emphasis on enabling state management in CICS Web-enabled applications.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP cookies</strong></td>
<td>• State can be preserved even if browser session terminates.</td>
<td>• User must have cookies enabled in their browser.</td>
</tr>
<tr>
<td></td>
<td>• Requires no additional configuration of WebSphere AS.</td>
<td>• Can be rejected by browsers.</td>
</tr>
<tr>
<td></td>
<td>• Can preserve state between browser sessions.</td>
<td>• Large amounts of data may need to be passed from server to browser, unless combined with unique session-IDs.</td>
</tr>
<tr>
<td></td>
<td>• Can store all state data in cookie, making it easy for workload balancing across multiple servers.</td>
<td></td>
</tr>
<tr>
<td><strong>URL rewriting</strong></td>
<td>• Does not require any special browser or server function.</td>
<td>• State data is immediately revealed by viewing the URL.</td>
</tr>
<tr>
<td></td>
<td>• Can store all state data in URL, making it easy for workload balancing across multiple servers.</td>
<td>• Each URL request must store the persistent data in its parameter list.</td>
</tr>
<tr>
<td></td>
<td>• Cannot be rejected by a browser.</td>
<td>• A failed HTTP request will lose the data.</td>
</tr>
<tr>
<td><strong>WebSphere sessions</strong></td>
<td>• Can use cookies or URL re-writing.</td>
<td>• Cannot preserve state between browser sessions.</td>
</tr>
<tr>
<td></td>
<td>• Relatively easy to implement using servlet API.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Only small amounts of data passed from server to browser as state tokens.</td>
<td></td>
</tr>
<tr>
<td><strong>Hidden HTML form fields</strong></td>
<td>• Does not require any special browser or server functionality.</td>
<td>• State data can be viewed by looking at the HTML source.</td>
</tr>
<tr>
<td></td>
<td>• Does not require any special browser or server function.</td>
<td>• Each HTML form must store the persistent data as hidden fields.</td>
</tr>
<tr>
<td></td>
<td>• User unaware that state data is stored in the HTML form, although user can view this data.</td>
<td>• Cannot preserve state if browser session terminated.</td>
</tr>
</tbody>
</table>
Part 2. The think tank — designing the solution

In this part we walk through each of the CICS Web-enablement solutions. For each scenario, we discuss the different ways of designing workload managed solutions in order to provide failover and load balancing.
Chapter 4. Workload management for CICS Web support

In this chapter we discuss the different workload management techniques that can be used to facilitate the effective balancing and distribution of a CICS Web support (CWS) workload. These techniques are examined as they apply to a CWS direct connection and to the CICS WebServer Plugin architectures.

First, we examine the usage of these techniques with a Web-aware CICS application design, in which the HTTP-aware presentation logic is integrated within the CICS program. Then, we analyze some Web-enabled 3270 applications.

4.1 Web-aware applications

A CICS Web-aware application is one that contains the HTTP based presentation logic necessary for sending output to a Web browser. For further details on using CWS, refer to 1.2.1, “CICS Web support” on page 6.

The four principal technologies that can be used to perform workload balancing with CWS Web-aware applications are these:

- OS/390 Communications Server (TCP/IP)
- OS/390 Web server and the EXCI (with the CICS WebServer Plugin)
- CICS dynamic routing of program LINK requests (in conjunction with CICSPlex SM and MVS WLM)
- File sharing

Figure 24 illustrates how each of these techniques apply, both when using a CWS direct connection, and when using the CICS WebServer Plugin.

![Figure 24. CWS Web-aware workload management options](image)
4.1.1 OS/390 Communications Server

In this section we discuss the different mechanisms available in OS/390 Communications Server for workload balancing incoming IP requests across multiple CICS regions or Web server address spaces.

TCP/IP port sharing

TCP/IP port sharing is a feature of OS/390 Communications Server, that allows multiple address spaces on the same LPAR to listen for requests on the same TCP/IP port. This allows for incoming work to be distributed across multiple CICS regions in the same LPAR, all of which would need to have a TCPIPSERVICE configured to listen for requests on the same port.

TCP/IP port sharing can be utilized to distribute an incoming TCP/IP workload over multiple cloned CICS listener regions. This can be used to alleviate the situation where a single CICS region becomes CPU constrained (that is, it tries to consume more than 100% of a single CPU in a multi CPU LPAR).

Figure 25 illustrates how TCP/IP port sharing spreads incoming IP requests across two CICS listener regions. Note that it is also possible to use this technique with the OS/390 Web server. For further details, refer to 2.2.1, “Port sharing” on page 33.
DNS connection optimization
DNS connection optimization is a feature of IBM OS/390 Communications Server V2.5, and uses the function of the dynamic DNS server and MVS WLM together, to balance TCP/IP requests across different TCP/IP hosts, in effect allowing requests to be workload balanced across different OS/390 LPARs (unlike TCP/IP port sharing). Since it relies on the OS/390 DNS server, it will only function if the IP requests use hostnames, and not IP addresses. Its usage in doing workload balancing for IP requests in a sysplex is illustrated in Figure 26.

**Note:** For further details, refer to 2.2.2, “DNS connection optimization” on page 34.

Sysplex Distributor
Sysplex Distributor is a feature of OS/390 Communications Server V2R10. It provides for balancing of IP packets across multiple OS/390 IP stacks. Similarly to the function of DNS connection optimization it allows IP requests to be balanced across different LPARs. It does, however, have several advantages, in that it does not work at the hostname level, and is also closely integrated with OS/390 dynamic virtual IP address (VIPA) support. For further details, refer to 2.2.4, “Sysplex Distributor” on page 37. Its usage in workload balancing IP requests in a sysplex is illustrated in Figure 26.

![Figure 26. TCP/IP workload management in a sysplex](image-url)
4.1.2 The OS/390 Web server

A CWS Web-aware application can be invoked using the OS/390 Web server, as opposed to using the services of the CICS TCP/IP listener. In this case, the CICS WebServer Plugin runs within the Web server and routes requests to the CICS region by means of the External Communications Interface (EXCI). This CICS region to which the requests are routed can be in any connected CICS system in the sysplex, since the EXCI requests can be routed via the S/390 coupling facility. For further details on the CICS WebServer Plugin, refer to 1.2.1, “CICS Web support” on page 6.

One point worth mentioning at this time is the possible benefit that utilizing the CICS WebServer Plugin provides in a situation where the CICS region is CPU constrained. By utilizing the CICS WebServer Plugin, a significant proportion of the Web processing will be off-loaded from the CICS region to the Web server address space. This is likely to increase the overall CPU requirements for the workload. However, it has the advantage that less CPU will be consumed within the CICS region, but instead, can be spread across the Web server address space, and thus can be distributed across LPARs in the sysplex.

Cloning Web servers

A simple and effective way of workload balancing across multiple Web servers is to use TCP/IP port sharing, which allows multiple Web servers in the same LPAR to listen for requests on the same port. This provides for greater scalability than can be provided by one Web server address space, and allows for failover if one address space terminates unexpectedly. This is illustrated in Figure 27.

Figure 27. TCP/IP port sharing — OS/390 Web server
Scalable mode
The OS/390 Web server can be configured to run in scalable mode. In this mode the OS/390 Web server operates in conjunction with MVS WLM running in goal mode to balance work across a group of Web server address spaces. The work is divided across multiple address spaces, namely the queue manager and one or more queue servers.

This is similar in many ways to simple cloning and the use of TCP/IP port sharing, but offers the advantage that it allows you to separate the different type of work (such as servlets and GWAPI programs) into different queue server address spaces, to which you can assign different WLM goals. Also, it is possible to configure additional queue server address spaces to be started as required, in order to increase the capacity when required. For further details, refer to 2.5.2, “WebSphere on OS/390” on page 44.

SSL considerations
The OS/390 Web server is capable of handling SSL encrypted HTTP sessions. If you use this facility you should be aware that the SSL handshake process may considerably increase the CPU utilization of the Web server address space. This CPU utilization can be significantly decreased by using persistent HTTP connections (controlled by the MaxPersistRequest and PersistTimeout directives); and by enabling SSL session-ID re-use (controlled by the SSLV2Timeout and SSLV3Timeout directives).

You should also consider that the use of the CICS WebServer Plugin can off-load a significant amount of CPU processing from the CICS listener regions(s). This can be significant when a single CICS region is approaching the maximum capacity of a CPU. Utilizing the OS/390 Web server and the CICS WebServer Plugin moves the CPU requirement for processing HTTP requests, to the HTTP server, where the architecture is more suited for parallel processing on multiple CPUs.

For further information on configuring SSL with the OS/390 Web server, refer to HTTP Server Planning, Installing, and Using, SC31-8890; and to OS/390 e-business Infrastructure: IBM WebSphere Application Server 1.2 Customization and Usage, SG24-5604.
EXCI
The EXCI is used by the CICS WebServer Plugin when requests are sent to an attached CICS region. The EXCI uses the CICS MRO protocol, and so is capable of using the cross system coupling facility (XCF) to send requests to any connected CICS region in a Parallel Sysplex. In addition the EXCI supplies a user exit (DFHXCURM), which can be used to change the destination APPLID in the EXCI request. However, it is important to note that the CICS WebServer Plugin uses the EXEC CICS version of the EXCI interface as opposed to the Call version (which is used by the OS/390 CTG). The important difference here is that when the EXCI returns a retryable error, the EXCI itself will automatically retry the EXCI call up to five times when using the Call interface. For more information on the EXCI, refer to 5.1.3, “EXCI” on page 82, and to chapter 8 in the CICS External Interfaces Guide, SC33-1944.

If you wish to do workload balancing for your EXCI requests from the CICS WebServer Plugin across several CICS listener regions, then you can use logic in the EXCI user exit (DFHXCURM) to balance requests across available CICS regions. We suggest two different strategies for doing this in our discussion of the usage of the OS/390 CTG (see 5.1.3, “EXCI” on page 82), using either a region health table in the DFHXCURM, or the state data facilities provided by CICSPlex SM.

---

EXCI pipe limits

A single OS/390 address space is limited (by the CICS MRO code) to only being able to allocate up to 100 pipes (sessions). In a CICS system the maximum pipe usage is limited only by the number of sessions defined in the CICS SESSIONS definition, which can be up to 999. Thus a Web server using the CICS WebServer Plugin to send requests to a CICS region will be limited to handling only 100 simultaneous EXCI requests to all attached CICS regions. However, the maximum number of parallel HTTP requests that the CICS WebServer Plugin can serve will be several-fold greater than this, since each EXCI pipe is only allocated for a short duration during the call to CWS.
4.1.3 CICS dynamic routing

CICS multi region operation (MRO) is a widely used technique that is a central part of CICS scalability. The modifications to the dynamic routing program, available in CICS TS V1.3, expand on this capability, allowing CICS LINK commands to be dynamically routed to any connected AOR (Figure 28).

Cloning listener regions
As illustrated in Figure 26, the capabilities of OS/390 Communications Server allow IP requests to be balanced across either multiple address spaces in the same LPAR (TCP/IP port sharing) or across different LPARs in the same sysplex (DNS connection optimization or Sysplex Distributor).

Dynamic routing of LINK requests
The enhancements to the dynamic routing in CICS TS V1.3 allow LINK requests to be dynamically routed from the listener region, to an AOR. This can be used in conjunction with CICSPlex SM, and the MVS Workload Manager to dynamically route requests across a series of cloned AORs based on load in the AORs. For further details on workload management with CICSPlex SM, refer to 2.3.1, “CICSPlex SM” on page 39.

Managing state data
If you clone your CICS listener regions, you will need to consider how to share any application state across the cloned CICS regions. This is because once you use an IP spraying technology (such as TCP/IP port sharing or Network Dispatcher) there is no way of determining to which particular CICS region any individual request will go. For more details, refer to Chapter 3, “Matters of state” on page 49.
SSL considerations
As of CICS TS V1.3, the CICS TCP/IP listener is capable of supporting SSL HTTP sessions, as well as normal unencrypted HTTP sessions. To establish an SSL session, the client (Web browser) and the server must participate in a flow of requests known as an SSL handshake. There are three distinct types of the SSL handshake:

1. **Full handshake** — This will be performed when the client initially establishes the SSL connection, since there is no session ID in the client hello message. A full handshake will also be performed when the server decides a submitted session ID is not valid for re-use.

2. **Resume session** — This operation will be performed when the client establishes an SSL connection that includes a session ID for the session to be resumed, and the server decides it is a valid session to be resumed.

3. **No handshake** — This will be performed when a new SSL request is received from a Web browser via a previously established persistent HTTP connection.

A schematic logic diagram of this process is given in Figure 29.

---

**Figure 29. Types of SSL handshake**
Thus when an SSL session is established with a CICS region, an affinity is established with one particular region, since the SSL session-ID is cached within the CICS region's address space. This means that if a subsequent HTTP request from the same client is routed to a different CICS region, the required SSL session-ID will not be found in that region's cache, and so a new SSL handshake will again have to be performed (Figure 30).

This is detrimental to performance, since the SSL handshake is an expensive operation, and can in fact cost more CPU cycles than the transmission of the data itself. Note that the S/390 Cryptographic Coprocessor Feature is a unique feature of S/390 that is capable of significantly reducing the CPU cost of both SSL handshakes and data encryption. For a detailed performance study of CWS and SSL usage, refer to *A Performance Study of Web Access to CICS*, SG24-5748.

The simplest means of ensuring that such a situation does not occur is to utilize HTTP persistent connections. An HTTP persistent connection is one which does not immediately close the underlying IP socket after each HTTP request. If a SSL session is used over such an HTTP persistent connection, no SSL handshake will be required until the another socket is required to be opened. In addition, a subsequent HTTP request over such a persistent HTTP connection will be guaranteed to go back to the initial region with which the socket was opened. If SSL connections are used within CICS, it is advisable that a dedicated set of regions is used to handle these requests, due to the CPU intensive nature of SSL requests and the benefits of servicing subsequent SSL requests in the same region.
A second consideration, when using SSL sessions, is that the maximum number of CICS S8 TCBs which handle SSL sessions has a finite limit. This is due to storage consumed below the 16 MB line by each TCB. Depending on the region's requirements for "below the line" storage, this will restrict the maximum number of TCBs to a few hundred. Since a persistent HTTP connection has an affinity with an individual S8 TCB, if an additional SSL session is requested with a CICS region, and all the S8 TCBs are already in use, then one of the existing S8 TCBs in use will be "stolen" by the new SSL session. If the previous Web browser, that had its TCB stolen then sends a subsequent SSL request, it will then steal back another TCB, and thus will itself have to perform an additional SSL handshake, even if previously it was using a persistent HTTP connection.

You can see that this situation will significantly increase the number of SSL handshakes required to maintain a population of Web browsers. For this reason it is advisable to limit the number of SSL sessions in use at one time with one individual CICS region, to be less than or equal to the number of S8 TCBs.

**APAR PQ31399**

If using the SSL session support with CICS TS V1.3, you should apply the fix for APAR PQ31399 and associated prerequisites. This enables SSL session ID re-use across different TCBs in a CICS region.

### 4.1.4 File sharing

File sharing is an important feature of a CICS workload management solution, since it allows multiple access to the same file from many different CICS AORs. As of CICS TS V1.1, VSAM RLS is now the primary method of choice for allowing file sharing in a sysplex environment, and is described further in 2.3.3, “VSAM RLS” on page 41. Note that DB2 or IMS data sharing could also be utilized for the same purpose, as can CICS Coupling Facility Data Tables.
4.2 3270-based applications

When accessing 3270 transactions using CICS Web support, the facilities of the 3270 Web bridge are used to provide an interface between the HTML based Web world and the 3270 based CICS transactions. For an introduction to the 3270 Web bridge, refer to 1.2.1.3, “3270 Web bridge” on page 12.

State management with the 3270 Web bridge
State data for a 3270 Web bridge transaction is stored in the client HTML form date by utilizing hidden HTML form fields, and in CICS storage that is unique to a specific region. Transactions that arrive without these hidden form fields are assumed to be new transactions by the 3270 Web bridge exit. Transactions that contain the hidden state data token are passed along to the bridge exit as existing transactions. This means it is not possible to devise a mechanism in CICS to share the state data used by the 3270 Web bridge across different CICS regions, as can be done when using a Web-aware CWS application.

Figure 31 shows an example of the flow of a 3270 Web bridge transaction. Of particular importance is the initial call to the 3270 Web bridge exit where the GETMAINed storage for the bridge facility and state data is obtained. As shown in this diagram, the creation of the bridge facility and state data in CICS storage has created an affinity to this particular region.

Figure 31. State management for 3270 Web bridge

Figure 31 also highlights another important point as it pertains to the 3270 Web bridge and workload management. Since 3270 Web bridge transactions are not associated with “real” 3270 terminals, a pseudo-terminal is created by
the 3270 bridge for these transactions. The pseudo-terminal and the other elements required to enable the 3270 Web bridge support to function are called a bridge facility. This bridge facility is unique to the CICS region in which it was created, and unlike a real 3270 terminal definition, cannot be shipped to another region, meaning that the 3270 transaction running under the bridge, cannot be transaction routed to an AOR.

As a result of these considerations, the choice of technologies that can be used to perform workload management with 3270 based applications is more limited than with Web-aware applications. Nevertheless, we will describe the possibilities that do exist. They are as follows:

- Communications Server (if using the CICS WebServer Plugin)
- OS/390 Web server
- CICS dynamic routing of program LINK requests
- File sharing

Figure 32 illustrates how each of these techniques apply both when using a CWS direct connection and when using the CICS WebServer Plugin.

![Figure 32. CWS 3270 Web bridge workload management overview](image)

### 4.2.1 OS/390 Communications Server

Since the state of the 3270 bridge facility is stored within the region that the 3270 Web bridge runs in, it is not possible to use any of the workload balancing features of OS/390 Communications server (such as TCP/IP port sharing, or DNS connection optimization) when using the 3270 Web bridge with a direct connection to CICS Web support. However, it is still possible to
use the workload balancing features with the OS/390 Web server, if this is used to pass requests to CICS via the EXCI.

4.2.2 OS/390 Web server

The CICS WebServer Plugin can be used in conjunction with the OS/390 Web server to send request to the 3270 Web bridge. However, although multiple Web server address spaces can be used, and requests spread across them using TCP/IP port sharing, or Sysplex Distributor, they must still all go to the same CICS listener region, due to the restriction in sharing state when using the 3270 bridge.

Since in this situation the OS/390 Web server must send all EXCI requests to the same CICS listener region, it is not possible to use any form of workload balancing in the EXCI user exit (DFHXCURM). Doing so would cause intermittent failures in the running of a conversational transaction, due to the loss of state and of the lack of a bridge facility.

4.2.3 CICS dynamic routing

It is not possible to define the transaction that runs under the 3270 bridge as a candidate for transaction routing. That is the transaction cannot be defined as remote, either statically or by use of dynamic transaction routing. However, it is still possible to dynamically route any EXEC CICS LINK calls that are made by the 3270 application, on to an attached AOR, providing, however, that the linked-to program obeys the DPL subset rules (in particular, it performs no terminal operations).

Since usage of the 3270 Web bridge itself can be fairly CPU intensive you should be sure that your CICS region does not become CPU constrained when peak loads are experience. In such a situation some temporary relief my be found by using the CICS WebServer Plugin to off-load some of the processing to the OS/390 Web server, but you may still find that CPU usage becomes an issue. If so, you should consider redesigning your application with separate business and presentation logic, which will enable you to choose from a range of workload management technologies and also from a much wider choice of Web-enablement techniques. For a comparison of the CPU usage of the 3270 Web bridge, and other CICS Web-enablement techniques, refer to A Performance Study of Web Access to CICS, SG24-5748.
4.2.4 File sharing

File sharing is an important feature of a CICS workload management solution, and can still be utilized with the 3270 Web bridge. However, since it is not possible to balance Web requests via the 3270 Web bridge across multiple AORs, there is little to be gained from the technique of file sharing, from a workload balancing point of view.
Chapter 5. Workload management for the CTG

In this chapter we discuss the different techniques available for Workload management when using the CICS Transaction Gateway (CTG). We first consider the CTG on OS/390, and then the CTG on distributed platforms, since the technologies differ considerably. For an introduction to the usage of the CTG for Web access to CICS, refer to 1.2.2, “CICS Transaction Gateway” on page 14.

5.1 CTG on OS/390

In this section we describe the various workload management configurations possible when using the CTG on OS/390.

The five principal technologies that can be use to perform workload balancing as part of this configuration are as follows:

- OS/390 Communications Server (TCP/IP)
- WebSphere Application Server
- The External CICS Interface (EXCI) as used by the CTG
- CICS dynamic routing of program LINKs (in conjunction with CICSPlex SM and MVS WLM)
- File sharing

The relationship between these technologies is illustrated in Figure 33. We will consider each in turn, and discuss what workload management technologies are available at each point.

---

Figure 33. CTG OS/390 workload management options
5.1.1 OS/390 Communications Server

In this section we discuss the different mechanisms available for workload balancing IP requests using IBM OS/390 Communications Server.

TCP/IP port sharing
TCP/IP port sharing, a feature of OS/390 Communications Server, allows two different address spaces on the same OS/390 LPAR to listen for requests on the same TCP/IP port, and balances requests across them. When using the OS/390 CTG this allows either multiple WebSphere AS address spaces (if using servlets), or multiple CTG Java gateway applications (if using applets), to all listen for requests on the same single TCP/IP port. This gives a single system view, and also provides for a higher degree of scalability than one address space can offer. (See Figure 34.)

For a discussion of the performance benefits of TCP/IP port sharing when used with the CTG Java gateway application, refer to *A Performance Study of Web Access to CICS*, SG24-5748. For further details about OS/390 Communications Server and TCP/IP port sharing, refer to 2.2.1, “Port sharing” on page 33.

DNS connection optimization and Sysplex Distributor
It is not possible to use the DNS connection optimization feature of Communications Server to balance work across different Web servers or CTG Java gateway application running on different LPARs, since this feature is not supported by these servers. Instead it is possible to use Sysplex Distributor (available in OS/390 V2.10) to balance incoming IP request across different IP addresses. For further details about Sysplex Distributor, refer to 2.2.4, “Sysplex Distributor” on page 37.
5.1.2 WebSphere AS

On OS/390, WebSphere Application Server (WebSphere AS) provides both the Web server (IBM HTTP Server) and the associated servlet engine. There are two very different ways of scaling Web servers, as described in the following sections.

Cloning Web servers

A simple and effective way of workload balancing using the OS/390 Web server is by simple cloning of the Web servers. These clones can either run on the same LPAR and use TCP/IP port sharing to do workload balancing for the incoming IP requests across them. Alternatively, a technique such as Sysplex Distributor or Network Dispatcher can be used to distribute work across multiple Web servers running in different LPARs. Furthermore, the two techniques can be combined together, as illustrated in Figure 35.

Note that if Web server address spaces are cloned, you will need to consider the implications of such a configuration on the handling of state by your Web application. If your servlet application stores state internally within the servlet, either using WebSphere sessions or by other means, this must be capable of being shared across the cloned Web server. For a full discussion on the different alternatives of managing state data in a CICS Web application, refer to Chapter 3, “Matters of state” on page 49.
Scalable mode
On OS/390 the Web server has a mode of operation referred to as scalable mode, in which a queue manager address space interfaces with MVS WLM to balance work across multiple queue server address spaces. This can be used in a fashion similar to TCP/IP port sharing to balance work across multiple address spaces in the same LPAR. For further details, refer to 2.5.2, “WebSphere on OS/390” on page 44. Its primary advantage is that of being able to interface with MVS WLM using goal mode, but it is somewhat more complicated to set up and configure.

5.1.3 EXCI
The EXCI is used by the OS/390 CTG to send requests to the CICS region specified in the ECI request. To make an ECI call, the OS/390 CTG uses the EXCI CALL interface. The EXCI CALL interface provides a low level API for calling a CICS program using a COMMAREA, and consists of the following six commands:

1. Initialize_User
2. Allocate_Pipe
3. Open_Pipe
4. DPL_Request
5. Close_Pipe
6. Deallocate_Pipe

However, as of V3.1.2 the CTG does not perform a Deallocate_Pipe after it performs the initial ECI request from any given thread. This means that a pipe stays allocated for the lifetime of the thread and is not deallocated until one of the following events occurs:

- The Web server (or CTG address space) terminates.
- The CICS region terminates.
- The CICS connection is placed out of service or IRC is closed.
- An EXCI retryable response occurs.

An EXCI retryable response is a specific kind of error meaning that the request can be re-issued. EXCI retryable responses are defined in Table 3.
Table 3. EXCI retryable responses

<table>
<thead>
<tr>
<th>EXCI response</th>
<th>EXCI command</th>
<th>ECI return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_CICS_IRC_STARTED</td>
<td>Initialize_User</td>
<td>-3 (ECI_ERR_NO_CICS)</td>
</tr>
<tr>
<td>NO_CICS</td>
<td>Open_Pipe</td>
<td>-3 (ECI_ERR_NO_CICS)</td>
</tr>
<tr>
<td></td>
<td>DPL_Request</td>
<td></td>
</tr>
<tr>
<td>NOPIPE</td>
<td>Open_Pipe</td>
<td>-16 (ECI_ERR_RESOURCE_SHORTAGE)</td>
</tr>
</tbody>
</table>

When a retryable response occurs, the EXCI pipe will be de-allocated by the CTG, and then re-allocated when the next ECI request is made.

**EXCI pipe limitations**

A single OS/390 address space is limited (by the IRC code) to only being able to allocate up to 100 EXCI pipes in total, to all attached CICS regions. In contrast, maximum pipe usage in a CICS region is limited only by the number of sessions defined in the CICS MRO sessions definition, which can be up to 999. Thus the following rules apply to the use of the EXCI by the CTG from a servlet:

- The Web server directive `MaxActiveThreads` sets the maximum number of concurrent requests that can be serviced. This is not the same as the maximum number of users, since the OS/390 Web server will queue requests to use threads, according to the setting of the TCP/IP `SOMAXCONN` parameter or the Web server `ListenBackLog` directive.

- Each thread that services an ECI request will allocate an EXCI pipe. This pipe remains allocated until either a failure occurs or the CICS MRO connection is terminated.

- Such pipes will be re-used by subsequent requests to the servlet, but pipes cannot be shared across different Web server threads.

If the number of threads using the CTG exceeds 100, the 101st EXCI `Allocate_Pipe` call will fail with a SYSTEM_ERROR response, and a reason code 608; the ECI application will receive a return code -9 (ECI_ERR_SYSTEM_ERROR). Since this means a limitation in the sending address space has been reached, you should consider configuring `MaxActiveThreads` in your Web server to be less than or equal to 100.

If the CTG tries to allocate more pipes than there are available sessions defined in the CICS sessions definition, the EXCI `Open_Pipe` will fail with a RETRYABLE response, and a reason code 202; the ECI application will receive a return code -16 (ECI_ERR_SYSTEM_ERROR) and DFHXCURM.
will be invoked. In this instance you should consider configuring the RECEIVECOUNT parameter in the CICS sessions to be at least 100.

If you decide that you require more than 100 threads in your Web server, we recommend that you do workload balancing across Web servers using TCP/IP port sharing, or utilize the Web server in scalable mode, which allows the usage of multiple HTTP Q server address spaces (each of which will be able to utilize 100 pipes with the CTG).

**DFHXCURM**

The EXCI user-replaceable module, DFHXCURM, provides for simple failover and round-robin workload balancing by providing a means to capture retryable errors and modify the destination APPLID specified in the EXCI call (see Figure 36). For further information on how we designed a workload balancing module, DFHXCURM, and the results of using it in our workload failover tests, refer to Chapter 9, “CTG servlets with WebSphere AS (OS/390)” on page 141.

**XCF group membership**

An important consideration when the CTG is used either by WebSphere or with the Java gateway application is the usage of slots in the DFHIR000 XCF (cross-system coupling facility) group in the sysplex couple data set. This XCF group is used by CICS when two address spaces on different LPARs communicate by means of MRO or EXCI. The maximum possible limit for group membership is 1023 members per group (it was raised from 511 by APAR OW21511), but the actual limit is defined in the MAXMEMBER parameter when the couple data set is formatted. To display this limit, the command `/D XCF,COUPLE` can be used.
If communication is between two CICS regions, then just one slot in this group is utilized for each MRO connection; however, when using the EXCI, one slot is used for each allocated EXCI pipe. You will remember that a CTG address space can allocate up to 100 EXCI pipes, and these pipes stay allocated for the lifetime of the CTG (the CTG Java gateway address space or the WebSphere address space). Therefore you should carefully monitor XCF group membership, if using EXCI XCF. The MVS command to monitor group membership is `$D XCF, GROUP, DPHIR000`.

For this reason, when using the CTG, it is advised that the CICS listener region should be situated on the same LPAR as the CTG. This will have the effect that EXCI XM (cross memory) will be used rather than EXCI XCF, and this is also likely to give better performance.

### 5.1.4 CICS dynamic routing and CICSPlex SM

CICS multi-region operation (MRO) is a widely used technique that is a central part of CICS scalability. The modifications to the dynamic routing program, available in CICS TS V1.3, expand on this, allowing CICS LINK commands to be dynamically routed to any AOR. As illustrated in Figure 37, this allows the CICS listener region specified in the APPLID on the EXCI call to function as a router region, and to use the facilities of the CICS dynamic routing to balance the EXCI request across a series of cloned AORs.

![Figure 37. MRO and dynamic routing](image)

### 5.1.5 File sharing

File sharing is an important feature of a CICS workload management solution, since it allows multiple access to the same file from many different CICS AORs. VSAM RLS is the primary method for allowing file sharing in a sysplex environment and is described further in 2.3.3, “VSAM RLS” on page 41. Also, DB2 or IMS data sharing could also be utilized to the same purpose, as can the usage of a CICS file-owning region.
5.2 CTG on distributed platforms

In this section, we describe the various workload management configurations possible when using the CTG on distributed platforms. The five principal technologies that can be used to perform workload balancing as part of this configuration are these:

- Network Dispatcher
- WebSphere Application Server
- CICS Universal Client Workload Manager
- CICS dynamic routing of program LINKs (in conjunction with CICSPlex SM and MVS WLM)
- File sharing

5.2.1 Network Dispatcher

Network Dispatcher is an advanced TCP/IP workload balancing product, capable of intelligently distributing incoming IP requests across a cluster of servers. Network Dispatcher itself runs on a PC or UNIX system (and also previously on a 2216 router) and can distribute requests across any number and type of connected servers. In our case the connected servers are likely to be Windows NT, AIX or Solaris machines, running either WebSphere AS or the CTG Java gateway application, but could also equally be OS/390 systems.

Using Network Dispatcher in this situation gives the advantage of providing higher throughput than can be achieved with one machine, and also allowing for failover of any given machine. On each server machine in the cluster, it is necessary to duplicate the configuration. This is illustrated in Figure 38 for a servlet configuration using WebSphere AS and the CTG, but would equally apply to an applet configuration using the CTG Java Gateway application. For an actual example of how we configured such a solution, refer to Chapter 10, “CTG servlets with WebSphere AS (Windows NT)” on page 173.

![Diagram](Image)

Figure 38. Network Dispatcher and WebSphere AS
Note that, in such a configuration, the presentation logic for your Web application will run within the servlet itself, and thus within WebSphere AS. Thus if you are considering such a servlet design, and considering cloning WebSphere Application Servers, you may need to consider a design that allows the sharing of state data across the WebSphere AS clones. For a full discussion on the different alternatives of managing state data in a CICS Web application, refer to Chapter 3, “Matters of state” on page 49.

Advisors
Network Dispatcher is a highly configurable product, and one of the means of tuning the workload balancing algorithm that it employs is to use its Advisor functionality. An Advisor is a lightweight client that runs as part of Dispatcher, but actually passes real commands to each server in order to determine the response time of each server. This response time is then added to the internal Dispatcher weights table to determine the routing of future requests. Several Advisors are supplied with Network Dispatcher including one for the HTTP protocol which is suitable for use with any Web server. Other sample advisors are supplied for HTTPS, FTP, NNTP, SMTP, POP3, IMAP, and Telnet protocols.

However, in addition to the sample advisors, a custom advisor may also be written. A custom advisor can be as complex or trivial as you deem necessary. A fully functional sample advisor servlet for use with WebSphere AS is provided with Network Dispatcher. This could be customized for use with CICS, such that a simple ECI request is made to call a null CICS program. This would provide a simple means of verifying the health of the CTG, the LU6.2 connection, and the CICS listener regions. Such an approach is illustrated in Figure 39.

Figure 39. Network Dispatcher and Java gateway application
For additional information on the operation of Network Dispatcher, refer to 2.6, “Network Dispatcher” on page 45.

5.2.2 WebSphere AS

In addition to Network Dispatcher, WebSphere AS itself provides several techniques for cloning and workload balancing across application servers. These can be used in conjunction with Network Dispatcher to provide a sophisticated workload management solution on distributed platforms. For further details, refer to 2.5, “WebSphere Application Server” on page 43.

5.2.3 CICS Universal Client Workload Manager

The CICS Universal Client on Windows NT incorporates its own Workload Manager. This allows simple round robin or a biased distribution, of ECI requests across configured CICS servers. This is illustrated in Figure 40.

![Figure 40. CICS Universal Client Workload Manager](image.png)

Workload management is configured on a per program basis. Both biasing and round-robin methods are able to provide failover. Should requests to a region unexpectedly fail, requests are directed to another region. The unavailable region is bypassed until a region time-out value is reached, at which point, if it is available, the region is returned to the pool of available regions. We recommend the use of the round-robin method, as it is a simpler scheme to configure, and it is not easy to calculate the relative weights to correctly use in a biased scheme.

Note that since it is not possible to determine the destination region when using the CICS Universal Client Workload Manager, the workload manager will only load-balance the initial flow from the addTerminal method when using the EPI, and the initial ECI request when using ECI extended logical units of work. If this were not the case, subsequent EPI requests could be sent to the wrong listener region, which would not have the correct terminal installed, and ECI requests could be sent to the wrong region, since all ECI requests in an extended logical unit of work must run within the same CICS region.
5.2.4 CICS dynamic routing and CICSPlex SM

CICS multi-region operation (MRO) is a widely used technique that is a central part of CICS scalability. The modifications to the dynamic routing program, available in CICS TS V1.3, expand on this technique, allowing CICS LINK commands to be dynamically routed to any attached AOR (or target region). As illustrated in Figure 41, this allows the CICS listener region(s) specified in the ECI request to function as router regions, and to use the facilities of the dynamic routing program in conjunction with CICSPlex SM to dynamically route the EXCI request to a series of AORs by means of a DPL call.

![Diagram of MRO and dynamic routing](image)

Figure 41. MRO and dynamic routing

5.2.5 File sharing

File sharing is an important feature of a CICS workload management solution, since it allows multiple access to the same file from many different CICS AORs. As of CICS TS V1.1, VSAM RLS is now the primary method of choice for allowing file sharing in a sysplex environment, and is described further in 2.3.3, “VSAM RLS” on page 41. Note that DB2 or IMS data sharing could also be utilized for the same purpose, as can CICS Coupling Facility Data Tables.
Chapter 6. Workload management for CICS CORBA client support

In this chapter we discuss the different workload management techniques that can be used to facilitate effective workload balancing with a CICS CORBA client solution. For an introduction to the usage of the CICS CORBA client support for Web access to CICS, refer to section 1.2.3, "CICS CORBA client support" on page 24.

The three principal technologies that can be used to perform workload balancing as part of this configuration are these:

- OS/390 Communications Server (TCP/IP)
- CICS dynamic routing of distributed program LINKs (in conjunction with CICSPlex SM and MVS WLM)
- File sharing

Figure 42 illustrates where each of these techniques apply to a CICS CORBA client scenario.
6.1 OS/390 Communications Server

In this section we discuss the different mechanisms available in OS/390 Communications Server for doing workload balancing on incoming IP requests across multiple CICS regions.

TCP/IP port sharing

TCP/IP port sharing is a feature of IBM OS/390 Communications Server, that allows multiple address spaces on the same LPAR to listen for requests on the same TCP/IP port. This allows for incoming work to be distributed across multiple CICS regions in the same LPAR, all of which would need to have a TCPIPSERVICE configured to listen for requests on the same port.

TCP/IP port sharing can be utilized to distribute an incoming TCP/IP workload over multiple cloned CICS listener regions. This can be used to alleviate the situation where a single CICS region becomes CPU constrained (that is, it tries to consume more than 100% of a single CPU in a multi CPU LPAR).

Figure 43 illustrates how TCP/IP port sharing spreads incoming IP requests across two CICS listener regions. For further details, refer to section 2.2.1, “Port sharing” on page 33.

![Figure 43. TCP/IP port sharing](image-url)
DNS connection optimization
DNS connection optimization is a feature of IBM OS/390 Communications Server V2.5. It uses the function of the dynamic DNS server and MVS WLM together, to balance TCP/IP requests across different TCP/IP hosts, in effect allowing requests to be workload balanced across different OS/390 LPARs (unlike TCP/IP port sharing). Since it relies on the OS/390 DNS server it will only function if the IP requests use hostnames, and not IP addresses. For further details, refer to section 2.2.2, “DNS connection optimization” on page 34. Its usage in workload balancing IP requests in a sysplex that is illustrated in Figure 44.

Sysplex Distributor
Sysplex Distributor is introduced with OS/390 Communications Server V2.10. It provides for balancing of IP packets across multiple OS/390 IP stacks. Similarly to the function of DNS connection optimization it allows IP requests to be balanced across different LPARs. It does, however, have several advantages, in that it does not work at the hostname level, and is also closely integrated with OS/390 Dynamic VIPA support. For further details, refer to section 2.2.4, “Sysplex Distributor” on page 37. Its usage in workload balancing IP requests in a sysplex is illustrated in Figure 44.

Figure 44. TCP/IP workload management in a sysplex
6.2 CICS MRO and dynamic routing

CICS multi region operation (MRO) is a widely used technique that is a central part of CICS scalability. The modifications to the dynamic routing program, available in CICS TS V1.3, expand on this allowing CICS LINK commands to be dynamically routed to any AOR. This can be used with the modular design of CICS CORBA client support to separate the CORBA function into the receiver function and the CICS Java function. This is illustrated in Figure 45.

The receiver function runs in the listener regions and is responsible for analyzing the inbound IIOP requests, invoking the DFHJIIOP program and marshalling and demarshalling the data, and returning the response to the caller.

The AOR function can be invoked in a separate region (or set of regions) by defining the DFJIIOP program as a remote program. This allows a DPL request to be made to the AOR for the program DFJIIOP, which then goes on to invoke the target CICS Java program. The Java program can then use the JCICS classes to LINK to existing logic in an existing CICS program, or to access CICS resources such as files or queues.

Figure 45. CICS CORBA client support and dynamic routing
CICSPlex SM and dynamic routing
This design can be used in conjunction with CICSPlex SM to dynamically route such DPL requests across a series of cloned AORs based on load in these AORs. For further details on workload management with CICSPlex SM, refer to section 2.3.1, “CICSPlex SM” on page 39.

However, there is also an opportunity for the dynamic routing request to be even more granular if required. As you will see in Figure 46, the CICS REQUESTMODEL definition is interrogated by the transaction CIOD prior to invoking the LINK to program DFJIIOP. One of the parameters of the REQUESTMODEL is the OMGMODULE field. The entry in this parameter defines a pattern that may match the qualified module name (coded in CORBA IDL), which defines the name scope of the interface and operation whose implementation is to be executed. This entry can be generic, so you could create two separate REQUESTMODEL definitions, for example, one specifying an OMGMODULE with a value of ‘A*’, the other specifying an OMGMODULE with a value of ‘B*’.

![Figure 46. CICS Native IIOP support: the role of the REQUESTMODEL](image)

The other parameter of importance in the REQUESTMODEL definition is TRANSID. The value entered in this parameter specifies the name of the CICS transaction to be executed when a request matching the specification of the request model is received.
So, if in the REQUESTMODEL definition relating to OMGMODULE A*, we specified a TRANSID of AIOD, and in the REQUESTMODEL definition relating to OMGMODULE B* we specified a TRANSID of BIOD, we have created a criteria for workload separation, based on the name of the input OMGMODULE.

Once again, with the careful use of the CICSPlex SM workload manager, we could direct all input modules whose names begin with A to be routed to one AOR (or set of AORs), and all input modules whose names begin with B to be routed to another AOR. In this scenario, all other input modules whose names begin with neither A nor B (and which would then use the default transaction CIOD) to be distributed evenly around all AORs by the CICSPlex SM workload manager. For this to work successfully, you would have to ensure that the definitions for transactions AIOD, BIOD and CIOD all specify PROGRAM DFHMIRS.

Sharing data areas over 32K
If the request or reply data being passed to DFJIIOP is less than 32K, the transfer of data from DFHJIIOP to the JCICS program is done using the traditional COMMAREA. However, if the data is greater than 32K, it is passed in a temporary storage 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 should be aware that if you are using distributed LINKs to DFJIIOP, you would need to define TSMODELs for prefixes DFIO and DFJO to ensure that these TS queues are sharable between the listener regions and the AORs. The recommended way to do this would be using a TS data sharing server to access a pool defined in a coupling facility.

SYNCPOINT considerations
Note that DFHIIOPA does not specify SYNCONRETURN on the EXEC CICS LINK request to DFJIIOP. This means that, after a distributed DPL to DFJIIOP, if the target Java program on the remote system links to another application program which issues an explicit SYNCPOINT, that SYNCPOINT request will fail. The APAR PQ37434 has been raised to address this issue and we recommend that you apply the available PTF. While this PTF changes DFHIIOPA so that the LINK to DFJIIOP specifies the SYNCONRETURN options, this change is intended to permit existing applications containing explicit SYNCPOINT requests to run in a dynamically workload balanced IIOP environment. We do not recommend that explicit SYNCPOINT requests should be added to new IIOP applications.
SSL considerations
As of CICS TS V1.3 the CICS TCP/IP listener is capable of supporting SSL encrypted HTTP sessions, as well as normal unencrypted HTTP sessions. Since there is a one to one affinity between each SSL session and a CICS S8 TCB, there are several important considerations if you wish to do workload balancing for SSL requests across multiple CICS regions. These considerations are detailed in “SSL considerations” on page 72 in Chapter 4, “Workload management for CICS Web support”.

File sharing
File sharing is an important feature of a CICS workload management solution, since it allows multiple access to the same file from many different CICS AORs. As of CICS TS V1.1, VSAM RLS is now the primary method of choice for allowing file sharing in a sysplex environment, and is described further in section 2.3.3, “VSAM RLS” on page 41. Also, DB2 or IMS data sharing could also be utilized for the same purpose, as can CICS Coupling Facility Data Tables.

6.3 Java considerations
When the CICS Object Request Broker (ORB) invokes the server object as part of the CICS transaction, the Java constructor is called first, and then the requested method. The method can call other methods, and after the last method is called, state data can be saved and the transaction ends. You should be aware that the method calls within each invocation of the Java constructor cannot themselves be routed. However, because subsequent calls to the Java constructor could be routed (as we discussed in section 6.2, “CICS MRO and dynamic routing” on page 94), you should consider saving the state data produced by method calls in shared temporary storage queues, which are available to all the AORs in the CICSPlex.

You should ensure that all AORs have access to either the MVS PDSE library SDFJLOAD which is maintained at a level compatible with the current release of the VisualAge for Java, Enterprise ToolKit for OS/390 (ET/390); or SDFJLOD1 which is maintained at a level compatible with Release 1. You will only require one of these libraries. You should choose the one that is compatible with the release of ET/390 that you are using, and the same library should be defined in all AORs. Additionally, each AOR should contain a reference to the PDSE library which holds the CICS Java server program objects that have been bound by ET/390.
Note that memory and CPU processing requirements to run CICS Java programs are significantly higher than for conventional CICS programs. Practically, the memory requirements can be addressed by increasing the allocated virtual storage in your Java AORs (using SIT parameter EDSALIM). The CPU processing requirements can also be significantly reduced by enabling the Java Hot Pooling feature, supplied in the fix to APAR PQ31328.

However, using the techniques described in this chapter for workload balancing can develop a truly scalable CICS Java solution within the existing bounds of your CICSPlex.
In this chapter we discuss the different workload management techniques that can be used to facilitate the effective workload balancing of Web requests to CICS when using IBM SecureWay Host On-Demand (HOD).

These techniques are examined in the following section as they apply to the following three components (Figure 47):

- Distribution of HOD applets from the *Web server* to HOD clients
- Workload balancing and failover of *Telnet servers*
- *CICS 3270* access

For an introduction to Web access to CICS using HOD, refer to 1.2.4, “Host On-Demand” on page 26. For in-depth information on using HOD, refer to IBM *SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing*, SG24-2149.
7.1 Web server

The focal point of the Host On-Demand solution is the Web server and HOD server configuration. Both of these components must reside on the same server. This can be an OS/390 LPAR, an AIX or Windows NT system, or a range of other systems.

There are several types of HOD clients; these are described further in 1.2.4, “Host On-Demand” on page 26. However, since the HOD client only needs to be downloaded, at the most, once per 3270 session, there is unlikely to be a need to do workload balancing for the download requests. Furthermore, HOD itself provides several types of clients such as the Function On-Demand client and the cached client which will also reduce the amount of data to be downloaded, and therefore the need to do workload balancing for the client download.

If the Web servers used by the HOD server are situated on OS/390, it would be possible to use the workload balancing techniques provided by OS/390 Communications Server, including TCP/IP port sharing, and Sysplex Distributor, as described in 2.2, “Communications Server” on page 33.

7.2 Telnet server

The telnet server is the principal component in the HOD Web solution. It is used to provide access to the client 3270 terminal running in the Web browser. It is the only component that is required to be active once the HOD client is downloaded from the Web server.

The most usual place for the telnet server, when using HOD for Web access to CICS, is on OS/390 using the 3270 capable telnet server (TN3270 server) provided by OS/390 Communications Server. This provides several workload balancing features which we will now discuss.

7.2.1 DNS connection optimization

The OS/390 TN3270 server implements DNS connection optimization and thus when coupled with the OS/390 dynamic DNS server, can provide effective load distribution across several TN3270 servers running in different OS/390 LPARs.

The OS/390 TN3270 server utilizes MVS Workload Manager (WLM) to register the availability of an open socket. This registration provides WLM with the information it needs to answer potential DNS requests for dynamic DNS updates. When the TN3270 server terminates, regardless of the reason,
it performs de-registration with WLM. Once TCP/IP has registered the
TN3270 server for connection optimization, the OS/390 DNS will request
updates from WLM at set intervals. The information returned by WLM is used
by the DNS to dynamically update the DNS data base. DNS connection
optimization can be utilized to distribute IP requests among different OS/390
LPARs in a sysplex. The OS/390 DNS is used to interrogate WLM for updates
(additions, deletions, and weighting changes) to the supplied static entries
that have been coded in the DNS data base (Figure 48).

![Figure 48. DNS connection optimization for TN3270 servers](image)

Figure 48 illustrates a functioning DNS connection optimization environment
where two OS/390 TN3270 servers have registered with WLM for DNS
connection optimization. In this example, the HOD client that attempts to
connect to the server TN3270E.ITSO.IBM.COM will be directed to either of the
TN3270 servers running on one of the two LPARs. The decision as to which
one to connect to will be determined by the OS/390 DNS based upon the
most recent weighting information obtained from WLM.
Workload balancing utilizing DNS connection optimization

In Figure 49 you can see that the Host On-Demand client is directed towards the TN3270 server with the highest WLM weighting information. As a reminder, be aware that the OS/390 DNS does not query WLM at each DNS invocation. DNS queries WLM at set intervals (default of sixty seconds) and continues to utilize this information until the next time it queries WLM. Therefore the DNS database may not accurately reflect the exact state of a failed address space, or, for the same reason, the addition of a new TN3270 server that has registered with WLM, until the next refresh interval is past.

Figure 49. HOD workload balancing with DNS connection optimization

TN3270 failover utilizing DNS connection optimization

Using DNS connection optimization provides for failover of a TN3270 server. In Figure 50 you can see an example of a failed Host On-Demand session, and how the auto-reconnect function of the Host On-Demand client will cause a second query to the DNS server for the `TN3270E.ITSO.IBM.COM` hostname. At this point, the DNS server will return another TN3270 server that is active, and the 3270 session will be re-established with this server.
Sysplex Distributor

Sysplex Distributor (in IBM OS/390 Communications Server V2.10) provides a means of allowing all the LPARs in the sysplex to be configured to give the appearance of a single multi-homed host, instead of independent TCP/IP stacks. Additionally, only one of the TCP/IP stacks will be responsible for interacting with IP clients. The other members of the Sysplex provide backup and communicate with the single TCP/IP region that controls routing to IP clients.

Using VIPA takeover and takeback, clients with cached IP addresses can be restarted on another TCP/IP stack that assumes control of the workload destined for the failed TCP/IP stack. Non-disruptive movement of server applications can also occur when used in conjunction with Sysplex Distributor. For further detail on Sysplex Distributor in OS/390 V2R10, refer to section 2.2.4, “Sysplex Distributor” on page 37.

7.2.2 Service Location Protocol (SLP)

Host On-Demand provides workload balancing and hot standby for TN3270 and TN5250 connectivity by providing Service Location Protocol (SLP) support in the HOD clients. To benefit from SLP, both the telnet server and clients must support SLP and be configured to participate. IBM Communications Server for Windows NT Version 6 and NetWare for SAA
SLP is defined in Request for Comments (RFC) 2165. It is a service-discovery method for TCP/IP-based communications, providing a simple and lightweight protocol for automatic advertisement and maintenance of intranet services and minimizing the use of broadcast and multicast in the network. SLP uses multicast, which targets a group of nodes, unlike broadcast, which targets all nodes. The benefit of multicast is that it sends one packet that all members of the group receive but that only the intended recipients read. A multicast packet is not isolated to a local segment; routers can forward it to whatever subnets are attached.

Specialized components called agents perform tasks and support services as follows:

**User Agent (UA)**
- This provides support for service query functions. It acquires/requests service information for user applications.

**Service Agent (SA)**
- This provides service registration and service advertisement.

**Directory Agent (DA)**
- This collects service information from Service Agents which is later requested by User Agents in intranets.

Services are described by the configuration of attributes associated with a type of service. A User Agent can select an appropriate service by specifying the attribute that it needs, in a service request. When the service request is returned, it contains a Uniform Resource Locator (URL) pointing to the service desired, and other information needed by the User Agent.

SLP can reduce overall network traffic by using scopes to manage client service requests. A scope is essentially a grouping method to organize servers into named groups. Scope values are defined by a network administrator, and may represent departments, regions or organizations. If desired, different scopes can be assigned for different services provided on the server. For additional information regarding SLP, please refer to *IBM Web-to-Host Integration Solutions*, SG24-5237.
Workload balancing utilizing SLP
To utilize SLP, the Host On-Demand client and the TN3270 server must be capable of supporting SLP and must be configured to use it. Figure 51 provides an overview of a Host On-Demand client obtaining recommendations from each of the TN3270 servers in the scope (HODSCOPE) that it is configured to use. A scope represents the "pool" of available Telnet servers that this Host On-Demand client can use.

In Figure 51, the Host On-Demand client has selected server TN3270A because it is the least loaded of any of the servers in the scope HODSCOPE. The load factor is a simple calculation of active TN3270 sessions divided by the available capacity; this gives a purely count-driven metric which is not based on CPU or memory utilization.

TN3270 failover utilizing SLP
Because SLP provides for the dynamic selection of a TN3270 server, it can be used to provide failover to TN3270 servers in the event a server fails. When the connection to TN3270A is no longer available, the auto-reconnect configuration option of the Host On-Demand client will cause SLP to locate another available server (if one exists). Assuming one exists, this new TN3270 server will be utilized.
Once a HOD client has connected to the telnet server, traditional CICS workload management facilities can be used to do workload balancing for the requests from the HOD clients.

The techniques of VTAM Generic Resource, CICS multi-region operation (MRO) and dynamic transaction routing, and VSAM RLS are illustrated in Figure 52. Each of these will be discussed in the order in which they play a role in workload management.
VTAM Generic Resources

VTAM Generic Resource provides a mechanism to distribute LU2 session establishment requests among the terminal-owning regions (TORs) that participate in the Generic Resources group.

When VTAM Generic Resources is utilized, there are two immediate benefits:

- **Workload distribution**: Sessions are distributed among TORs in the Generic Resources group.
- **Workload recovery**: If a member of the Generic Resources group fails, the Host On-Demand client can re-establish the connection using the same application name and be connected to another member of the Generic Resources group.

Figure 52 on page 106 shows a VTAM Generic Resource Group that could be made up of CICS terminal-owing region (TOR) configurations on each member of the sysplex. This means that even a failed OS/390 LPAR will not prevent clients from gaining access to surviving members of the sysplex.

Dynamic transaction routing

The dynamic routing function of CICS allows a set of CICS TORs to dynamically route requests across a set of application-owning regions (AORs). These TORs can be situated on any LPAR in the sysplex providing for load distribution across different LPARs in the sysplex. When used in conjunction with CICSPlex SM, requests from the TORs can be dynamically workload balanced across all the AORs. CICSPlex SM utilizes a variety of measurements to make routing decisions, and can be integrated with MVS Workload Manager (WLM) for this purpose.

File sharing

File sharing is an important feature of a CICS workload management solution, since it allows multiple access to the same file from many different CICS AORs. As of CICS TS V1.1, VSAM RLS is now the primary method of choice for allowing file sharing in a sysplex environment, and is described further in 2.3.3, “VSAM RLS” on page 41. Note that DB2 or IMS data sharing could also be utilized for the same purpose, as can CICS Coupling Facility Data Tables.
Part 3. The construction zone — building the solution

In this part we discuss three real-life scenarios in which we built workload management solutions for Web access to CICS. We describe the steps we had to take to configure each solution. Then we provide the results of our workload tests, together with details on failover in a variety of failure scenarios.
Chapter 8. CICS Web support

In this chapter, we document how we set up and configured a workload management solution using a CICS Web support (CWS) direct connection design. The following components were used to distribute work (Figure 53):

DNS connection optimization
DNS connection optimization was used to spread incoming TCP/IP requests across the two Web LPARs.

TCP/IP port sharing
TCP/IP port sharing was used to balance work across the two CICS listener regions in each Web LPAR.

CICSPlex SM and dynamic routing of LINK requests
The CICSPlex SM dynamic routing program was used to balance LINK requests from our CICS listener regions to our AORs.

VSAM Record Level Sharing (RLS)
We used the VSAM RLS feature to allow our two AORs to have shared access to the application’s VSAM files.

Figure 53. CICS Web support scenario
8.1 Configuring the scenario

We used seven CICS TS V1.3 regions spread across three separate LPARs in our OS/390 V2.8 Parallel Sysplex. Four CICS listener regions running on two separate Web LPARs were used to receive HTTP requests, this gave us failover at the CICS region and LPAR level. These listener regions routed requests onto two CICS AORs running in a separate target LPAR (note that a real-life configuration would be likely to use many more AORs than this, but we merely wanted to illustrate the different techniques possible). On each of the three LPARs, a CICSPlex SM Address space (CMAS) region was used to control the dynamic routing of CICS DPL requests from the listener regions to the AORs. We also used a CICSPlex SM Web User Interface (WUI) server to enable monitoring of the CICSPlex from a Web browser.

The application we used to demonstrate workload balancing in a CWS scenario was called Trader, a sample share trading CICS application. It used a CWS converter program called TRADERCV that performed the Web-aware HTTP presentation logic, and a business logic program called TRADERBL. It was configured so that the converter program could run in any one of the listener regions, and the business logic program in any one of the AORs. Incoming HTTP requests were distributed across the four listener regions using the DNS connection optimization and port sharing features of TCP/IP in Communications Server. The business logic program, TRADERBL, utilized two VSAM files, one for customer information, the other for company information. These were opened in RLS mode, enabling them to be shared between the AORs. This configuration is illustrated in Figure 54. For further details on the Trader application, refer to Appendix A, “The Trader application” on page 205.

![Figure 54. CWS Trader application](image-url)
8.1.1 Configuring TCP/IP

In order to implement TCP/IP port sharing and DNS connection optimization, we had to make several changes to the OS/390 TCP/IP configuration on the sysplex. We will describe these in the following section, but first let us look at the URL we used to invoke Trader:

```
```

This URL is made up of the following components:

- **tradcwxn** is the name of the MVS WLM group name registered with the OS/390 domain name server. Using this group name caused dynamic DNS address resolution to be invoked. (We will discuss how we configured this later in this chapter.)
- **8081** is the port number (used by all four listener regions) on which the CICS TCP/IP listener, receives HTTP requests.
- **tradercv** is the converter program Trader uses.
- **cwba** is the alias transaction under which the converter program runs.
- **traderbl** is the business logic program, the converter links to this.

### 8.1.1.1 DNS connection optimization

We utilized DNS connection optimization (Figure 55) to provide workload balancing across our two Web LPARs. (We would have preferred to have used the more advanced function available in Sysplex Distributor for this purpose, but it was not generally available at the start of this project.)

![Figure 55. DNS connection optimization configuration](image-url)
We will now provide an overview of the steps we took to configure TCP/IP. For further information, refer to the IBM Redbook, *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide, Volume 1: Configuration and Routing, SG24-5277*.

First we had to specify that the TCP/IP host was part of an MVS sysplex domain, and should communicate interface changes with the MVS WLM. This was enabled by adding a SYSPLEXROUTING entry to the TCP/IP stack profile member (TCPPROF), as shown in Figure 56.

```plaintext
; Control the IP layer
; IPCONFIG NODATAGRAMFWD SYSPLEXROUTING
```

*Figure 56. TCP/IP stack profile member showing SYSPLEXROUTING*

The name server ran with the started task name of NAMED and needed one port to be associated with it for both TCP and UDP sessions. We therefore amended the TCP/IP stack profile member with the PORT statement shown in Figure 57.

```plaintext
:PORT
  53 TCP NAMED ; OE Name Server
  53 UDP NAMED ; OE Name Server
; ENDPORT
```

*Figure 57. NAMED port allocation*

The last change to the TCP/IP profile member was to add an AUTOLOG statement for the NAMED started task, to enable automatic start-up of the name server, this is shown in Figure 58.

```plaintext
;; AUTOLOG the following servers
AUTOLOG 5
  NAMED ; Domain Name Server
ENDAUTOLOG
```

*Figure 58. AUTOLOG for NAMED procedure*

Next, we needed to specify the address of the host on which the local name server was running. In our case, the host was LPAR SC61 whose TCP/IP address was 9.12.2.25. This address was specified by coding a NSINTERADDR statement, in the TCPCDATA member of the parameter database, as shown in Figure 59. The presence of the NSINTERADDR
statement means that the local name server will be used to resolve names, rather than using site table lookup, which would normally be the case.

```
; NSINTERADDR 9.12.2.25
;
```

**Figure 59. TCPDATA member: NSINTERADDR statement**

Having configured TCP/IP, we then created the local name server procedure (called NAMED, as mentioned above), using the sample NAMED procedure as supplied in the SEZAINST dataset. The NAMED procedure we used is shown in Figure 60.

```
//NAMED PROC B='/etc/named.boot',P='53'
//******************************************************************************
//* 07-27-00 NAMED implemented to test DNS connection optimization for*  
//* Workload Management Redbook (SG24-6118) *
//*******************************************************************************
//NAMED EXEC PGM=EZANSNMD,REGION=0K,TIME=NOLIMIT,  
// PARM='POSIX(CN) ALL31(CN)/ -b &B -p &P -q -t 60 -l 0 -d 11'  
//* -d 11 for full debug  
//* -t 3600 to set wlm refresh to one hour  
//*STEPLIB DD DISP=SHR,DSN=TCP.SEZALINK  
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=F,LRECL=132,BLKSIZE=132)  
//SYSIN DD DUMMY  
//SYSERR DD SYSOUT=*  
//SYSCNT DD SYSOUT=*,DCB=(RECFM=F,LRECL=132,BLKSIZE=132)  
//CEEDUMP DD SYSOUT=*  
//******************************************************************************
```

**Figure 60. Local name server procedure, NAMED**

In Figure 60, notice the line which reads:

```
// PARM='POSIX(CN) ALL31(CN)/ -b &B -p &P -q -t 60 -l 0 -d 11'
```

This statement contained values which were applied to the domain data file. Of particular interest is the -l parameter, which had a value of 0. This causes the time to live (TTL) value to be set to zero, which means that every time this name server answers a query from a client workstation, it will instruct the workstation not to cache the answer. Some workstation TCP/IP implementations do not honor a TTL of zero. This can be a problem in the successful implementation of DNS connection optimization, since cached IP addresses will prevent work from being distributed across the different IP addresses.
We noticed that when we ran our workload tests that our Netscape and Internet Explorer browsers running on Windows NT did not honor this TTL of zero; instead, we had to wait for a few minutes before the resolved address was removed from the workstation cache.

Note also that, in Figure 60 on page 115, the NAMED procedure contained a cross reference to a boot file called /etc/named.boot, as shown in Figure 61.

```
;/etc/named.boot
; Type Domain Host or File
; directory /etc/dnsdata
; primary itso.ibm.com named.for cluster
; primary 2.12.9.in-addr.arpa named.rev
; primary 0.0.127.in-addr.arpa named.lbk
; cache . named.ca
; forwarders 9.12.2.7
; options query-log
```

*Figure 61. named.boot file*

The records in the boot file identified the type of name server, the zones over which it had authority, the location of data for setting up its name resolution database, and other configuration options. Notice the line containing the following information:

```
primary itso.ibm.com named.for cluster
```

In this line, the CLUSTER keyword was essential, because it indicated that this name server should query WLM for dynamic configuration updates. This was the key to a successful implementation of DNS connection optimization.

Note also the following two lines in Figure 61:

```
forwarders 9.12.2.7
directory /etc/dnsdata
```

The first line, forwarders, indicates that any queries which this name server cannot resolve should be forwarded to the site name server at address 9.12.2.7.
The second line, directory, points to the directory containing other configuration files required by this procedure. All of these files, and their purposes, are shown in Figure 62 through Figure 65.

The forward domain data file, named.for, is shown in Figure 62. This file contains address-to-name mapping information. The SOA (start of authority) record specifies the name server wtsc61oe, as the authoritative name server for the domain itso.ibm.com. The control entry $ORIGIN causes the string ‘itso.ibm.com’ to be appended to the following host names that do not end with a dot (‘.’). The NS (Name Server) record specifies the name server(s) in the zone, and the A (Address) records map the host names to the IP address of the network to which it was connected (for example, wtsc61oe was mapped to 9.12.2.25).

```
; /etc/dnsdata/named.for
;
$ORIGIN ibm.com.
itso IN SOA wtsc61oe.itso.ibm.com. ( 5 10800 3600 604800 0 )
;
   IN NS wtsc61oe
;
Owner   Class Type Record Data
localhost IN   A    127.0.0.1
wtsc61oe 0 IN   A    9.12.2.25
wtsc52oe 0 IN   A    9.12.2.17
```

Figure 62. Forward domain data file: named.for

The reverse domain data file, named.rev, is shown in Figure 63. This file contains IP address to name mapping information, and mapped IP addresses to hostnames in the network 9.12.2. For example, the PTR entry which starts with a ‘25’ maps IP address 9.12.2.25 to the host name wtsc61oe.itso.ibm.com.

```
; /etc/dnsdata/named.rev
;
$ORIGIN 2.12.9.in-addr.arpa.
@  IN SOA wtsc61oe.itso.ibm.com. wtsc61oe.itso.ibm.com. ( 2 10800 3600 604800 0 )
   IN NS wtsc61oe.itso.ibm.com.
;
17 IN   PTR wtsc52oe.itso.ibm.com.
```

Figure 63. Reverse domain data file: named.rev
The hints file, named.ca, is shown in Figure 64. This file contains the name and address of the authoritative root domain name server. The root domain name server contains the names of name servers in the top-level domains such as com or edu. The name server uses root server information when deciding which name server to contact when it receives a query for a host outside its zone of authority and it does not have the data in its cache.

```
./etc/dnsdata/named.ca
  . IN NS A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. IN A 9.24.104.109
```

*Figure 64. Hints file: named.ca*

The loopback file, named.lbk, is shown in Figure 65. This file contains a loopback address which a host would use to route queries to itself.

```
;/etc/dnsdata/named.lbk

; /etc/dnsdata/named.lbk
0.0.127.in-addr.arpa. IN SOA wtsc61oe.itso.ibm.com. wtsc61oe.itso.ibm.com ( 2 10800 3600 604800 00 )
0.0.127.in-addr.arpa. IN NS wtsc61oe.itso.ibm.com.
1.0.0.127.in-addr.arpa. IN PTR localhost.
```

*Figure 65. Loopback file: named.lbk*

Finally, we created a TCPIPSERVICE definition named DNS.TRAD in each of our listener regions. We purposely ensured that the name began with a ‘D’, as this is required for CICS to register with WLM for DNS connection optimization.

The MVS WLM registered group name is taken from a combination of part of the name of the CICS TCPIPSERVICE and the Web attach transaction defined in the TCPIPSERVICE. The characters of the TCPIPSERVICE name following the dot ‘.’ are concatenated onto the transaction identifier to form the group name. Thus, since we used the default Web attach transaction, CWXN, the group name generated was TRADCWXN. For more details on configuring the TCPIPSERVICE definition in CICS, refer to Figure 74 on page 125.

This new MVS WLM registered group name effectively becomes a new host name in the sysplex TCP/IP domain. As each of the four listener regions register with the same group name, WLM dynamically updates DNS to expand the list of available IP addresses for this group name. However, you should be aware that this dynamic update is only performed every 60 seconds, so potentially DNS might not be aware of any CICS region registering or deregistering for up to 59 seconds.
There were several ways in which we could have verified that DNS connection optimization was active; the two we chose are illustrated below. First, we configured our Windows NT workstation to use the new name server 9.12.2.25 as the first name server in its search order. Then we issued two consecutive `ping` commands, against the fully qualified groupname, and these returned alternate IP addresses for that group name, representing the two LPARs participating in the group; this is shown in Figure 66.

```plaintext
C:\>ping tradcwxn.itso.ibm.com
Pinging tradcwxn.itso.ibm.com [9.12.2.17] with 32 bytes of data:
  Reply from 9.12.2.17: bytes=32 time=80ms TTL=62
  Reply from 9.12.2.17: bytes=32 time=70ms TTL=62
  Reply from 9.12.2.17: bytes=32 time=110ms TTL=62
  Reply from 9.12.2.17: bytes=32 time=80ms TTL=62
C:\>ping tradcwxn.itso.ibm.com
Pinging tradcwxn.itso.ibm.com [9.12.2.25] with 32 bytes of data:
  Reply from 9.12.2.25: bytes=32 time=70ms TTL=62
  Reply from 9.12.2.25: bytes=32 time=60ms TTL=62
  Reply from 9.12.2.25: bytes=32 time=70ms TTL=62
  Reply from 9.12.2.25: bytes=32 time=70ms TTL=62
```

Figure 66. PING command output

Then we examined the job log of the participating CICS regions for the message shown in Figure 67. This message indicated that the region had registered with WLM to become part of the workload group.

```plaintext
DFHSO0115 08/08/00 14:20:18 SCSCPWA1 CICS has registered the group name TRADCWXN with Work Load Manager. The TCP/IP host name tradcwxn.itso.ibm.com will become available for DNS connection optimization.
DFHSO0107 08/08/00 14:20:18 SCSCPWA1 TCPIPSERVICE DNS.TRAD has been opened on port 08081 at IP address 9.12.2.25.
```

Figure 67. CICS message for DNS connection optimization

We found that although CICS registered with WLM, the name server would not resolve our hostname. After investigation, we found this was rectified by applying the fix for APAR PQ39719, and the associated fix for OW38086.
Because we were using DNS connection optimization, MVS WLM controlled which LPAR requests were to be routed to. Once the workload reached either LPAR SC52 or SC61, the CICS region in which it actually ran was determined by the port number in the URL. In our case, we specified port number 8081. However, because we had implemented TCP/IP port sharing, requests to this port were shared across the CICS listener regions. Port 8081 was shared by regions SCSCPWA1 and SCSCPWA2 on LPAR SC61, and also, by regions SCSCPWA3 and SCSCPWA4 on LPAR SC52.

Figure 68 shows the TCP/IP port sharing statements we added to our TCPPROF dataset on SC61 TCPIPOE.SC61.TCPPARMS(TCPPROF), to allow the regions SCSCPWA1 and SCSCPWA2 to participate in port sharing. We also made a similar change to the TCPPROF dataset on SC52 for the regions SCSCPWA3 and SCSCPWA4. Note that in our configuration we also were running two TCP/IP stacks (one for OS/390 and the other for UNIX System Services) and we had to make the changes shown in Figure 68 to both TCP/IP stacks.

```
PORT
  8081 TCP SCSCPWA1 SHAREPORT ; CICS SERVER
  8081 TCP SCSCPWA2 SHAREPORT ; CICS SERVER
```

Figure 68. TCP/IP SHAREPORT statement

Lastly we also added 08081 as the PORTNUMBER parameter in the CICS TCPIPSERVICE definition for all listener regions. By so doing, we created the situation where the port number 8081 (on each LPAR) was used by all four listener CICS regions. This allows TCP/IP to balance the workload across all four listener regions using a combination of DNS connection optimization and TCP/IP port sharing.

By implementing DNS connection optimization and TCP/IP port sharing we were expecting that the workload would be balanced across the entire sysplex. We used a variety of measurement criteria when the workload had completed to evaluate how successful this had been, and this is discussed in more detail in 8.2, “Running our workload” on page 131.

We noticed on certain occasions that TCP/IP port sharing was not being implemented properly. After investigation, we found that this was rectified by applying the fix for APAR PQ37993.
8.1.2 Configuring CICS

In order to configure our scenario we needed to make several changes to the CICS regions. The most significant CICS configuration task was to ensure that the SIT parameter DTRPGM (the dynamic routing program name) was set to EYU9XLOP so that CICSPlex SM dynamic routing of programs could be invoked. We discuss CICSPlex SM dynamic routing in more detail in 2.3.1, “CICSPlex SM” on page 39.

The remainder of the CICS resource definitions were for the configuration of the Trader application. Our CICS Web support Trader application consists of the CICS resource definitions listed in Table 4. For further details on how the Trader application functions, refer to Appendix A, “The Trader application” on page 205.

Table 4. CICS resources used by the Trader application

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>TCPIP SERVICE</th>
<th>TS MODEL</th>
<th>FILE</th>
<th>DOCTEMPLATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listener regions</td>
<td>TRADERCV</td>
<td>DNS.TRAD</td>
<td>WBSH</td>
<td>TRADBANN, TRADCMP, TRADEND, TRADFOOT, TRADHEAD, TRADQBS, TRADSIGN</td>
</tr>
<tr>
<td>AORs</td>
<td>TRADERBL</td>
<td></td>
<td>CUSTFILE, COMPFILE</td>
<td></td>
</tr>
</tbody>
</table>

Program definitions

Program TRADERCV is the converter routine explicitly coded in the URL. It is called on invocation to interpret the incoming HTML data, and is not eligible for dynamic program routing. Since this program was therefore static, we did not need to create a program definition for it, and instead utilized program autoinstall.

The program we needed to route was TRADERBL. In order to do so, we needed to install a definition of the program in the listener regions, with the ‘DYNAMIC’ option set to YES. This definition is displayed in Figure 69 through Figure 70. Since the program was static in the listener region, we did not supply a definition for it in those regions, but once again utilized program autoinstall.
Notice in Figure 70 that we did not specify the mirror transaction, leaving it to default to CSMI. This works fine when using CICSPlex SM dynamic program routing in “queue” mode. However, if we had wanted to use “goal” mode, we would have been required to specify a unique mirror transaction. See 9.1, “Configuring the scenario” on page 142 for more information on using “goal” mode and how to configure a CICS transaction in an MVS WLM service class.
File definitions
The VSAM file definitions for the files COMPFILE and CUSTFILE were defined in RLS mode to enable the files to be shared between the two AORs, SCSCPAA1 and SCSCPAA4. The COMPFILE is used by Trader to store a static list of company data, and the CUSTFILE used to store the number of shares held. The definitions for these files are shown in Figure 71 and Figure 72.

**Figure 71. RLS file definition for file CUSTFILE**

```
OVERTYPE TO MODIFY
CEDA Alter file( CUSTFILE )
File : CUSTFILE
Group : TRADERGL
DESCRIPTION -->
VSAM PARAMETERS
DSName --> CICSSYSF.SAMPLE.CUSTFILE.RLS
Password --> PASSWORD NOT SPECIFIED
Rlsaccess --> Yes
Lsrpoolid --> 1
READInteg --> Uncommitted
DSSharing --> Allreqs
Strings --> 010
Nsrgrpnum -->
REMOTE ATTRIBUTES
REMITName -->
REMOTE AND CFQATABLE PARAMETERS
* RECORDSIZE --> 1-32767
```

**Figure 72. RLS file definition for file COMPFILE**

```
OVERTYPE TO MODIFY
CEDA Alter file( COMPFILE )
File : COMPFILE
Group : TRADERGL
DESCRIPTION -->
VSAM PARAMETERS
DSName --> CICSSYSF.SAMPLE.COMPFILE.RLS
Password --> PASSWORD NOT SPECIFIED
Rlsaccess --> Yes
Lsrpoolid --> 1
READInteg --> Uncommitted
DSSharing --> Allreqs
STRINGs --> 001
Nsrgrpnum -->
REMOTE ATTRIBUTES
REMITName -->
REMOTE AND CFQATABLE PARAMETERS
* RECORDSIZE --> 1-32767
```
DOCTEMPLATE definitions

The HTML DOCTEMPLATEs were required by the converter program (TRADERCV), which only ran in the listener regions. Therefore the DOCTEMPLATE definitions for the HTML forms TRADBANN, TRADCOMP, TRADEND, TRADFOOT, TRADHEAD, TRADQBS, TRADSIGN, were installed only in the listener regions SCSCPWA1, SCSCPWA2, SCSCPWA3, SCSCPWA4. Figure 73 illustrates one of the DOCTEMPLATE definitions we used.

![Figure 73. DOCTEMPLATE definition](image)

You can see that all the templates were stored in the DFHHTML dataset, which in all our listener regions was the PDS CICSSYSF.APPL.TEMPLATE. You should be aware that access to this dataset can be a possible I/O bottleneck under heavy load. This can be avoided by storing the templates in memory. The best way to do this is by storing them as programs; details on how to do this are given in the IBM Redbook, *CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge*, SG24-5480.
TCPIPSERVICE definitions

In order for a CICS region to receive TCP/IP requests a TCPIPSERVICE had to be defined. A single TCPIPSERVICE was configured and installed on all the listener regions: SCSCPWA1, SCSCPWA2, SCSCPWA3, and SCSCPWA4. The definition is shown in Figure 74.

The TCPIPSERVICE name was “DNS.TRAD”. The “D” prefix signifies that we wished to register the TCPIPSERVICE to use DNS connection optimization, and the ‘.TRAD’ portion of the name indicated the first four characters of the DNS server name (in this case TRADCWXN). The interaction between CICS and MVS WLM when using DNS connection optimization is discussed in more detail in 8.1.1, “Configuring TCP/IP” on page 113.

We installed the same TCPIPSERVICE definition in all four listener regions. This meant that all four regions were listening on the same port (8081) and could also be addressed by the hostname “tradcwxn”. Note that, since the CICS regions SCSCPWA1 and SCSCPWA2 were running on separate LPAR to regions SCSCPWA3 and SCSCPWA4, port 8081 was not the same physical port for all four CICS regions. On SC61, SCSCPWA1 and SCSCPWA2 shared access to port 8081. On SC52, SCSCPWA3 and SCSCPWA4 shared port 8081.
SOCKETCLOSE and MAXTASK considerations

In our TCPIPSERVICE, we specified a SOCKETCLOSE value of 000010 (which represents 10 seconds). We had to experiment with this operand to get the workload to run successfully. We found that specifying a value of NO could sometimes cause our CICS region to hang, due to the fact that it became swamped with suspended Web attached CWXN tasks.

Once the number of tasks in a CICS region reaches the system max-task limit (200 in our region), no further transactions can be processed, and the region will hang. In a production environment, we highly recommended that you limit the maximum number of Web attach transactions that can be active. To do this, you need to add the CWXN transaction to a TRANCLASS with a MAXACTIVE setting equal to approximately half the region’s MXT value, to prevent your CICS region from hanging in such a manner.

We found that a SOCKETCLOSE value of a few seconds (we used 10) gave good results, since this allowed the socket to remain open until the Web driver sent its next request, but did not keep it open indefinitely. The recommendation would normally be that the SOCKETCLOSE value should be a small interval, slightly longer than the average Web user’s think time. This allows you to take advantage of the performance benefits of persistent HTTP sessions, but should prevent CWXN tasks from being indefinitely suspended in the CICS region.

Lastly, note that we specified a ‘BACKLOG’ value of 128, which indicates the number of input requests which will be buffered by TC/PIP. The value specified for this was equal to the value of the TCP/IP SOMAXCONN parameter, which we also set to 128. The SOMAXCON parameter specifies the maximum number of pending connection requests queued for any listening socket. It is recommended that the value you choose for BACKLOG should be less than or equal to SOMAXCONN. We found that if we let SOMAXCONN default to 10, our workload driver reported “early socket close” problems, and several transactions in a workload failed to run correctly.
TSMODEL definitions for storing state data

There are two sample CICS Web support state management programs, DFH$WBST and DFH$WBSR that are designed to handle state data. DFH$WBST uses EXEC CICS GETMAIN to allocate storage for saved state data, and DFH$WBSR saves state data in temporary storage (TS) queues. We initially used DFH$WBSR and made the underlying TS queues available in all the listener regions by defining a TS pool in the coupling facility. The TS model definition we used is shown in Figure 75. It was called WBSH, since this is the prefix of all the queue names used by CWS Trader application. The poolname TSQSPQA1 was the name of the temporary storage pool defined in the coupling facility. In order to use storage in this pool, we had to run a CICS TS pool server in each Web LP AR (SC61 and SC52). For details on how to configure a temporary storage pool server, refer to CICS Transaction Server for OS/390: Version 1 Release 3 Implementation Guide, SG24-5274, and Chapter 26 in the CICS System Definition Guide, SC33-1682.

Another issue we considered was how much space to allocate to the TS pool. In our case we defined a pool of 30MB, but we exhausted this space at one point during testing, and the result was that all the alias tasks went into a hung state because of the NOSPACE condition being returned on an EXEC CICS WRITEQ TS command. We rectified this situation by amending DFH$WBSR to handle the NOSPACE condition and to abend the offending task, rather than causing CICS to hang. We also took the additional measure of writing a short program started by the PLTPI, which browsed through all CICS TS queues, and deleted WBSH queue records with a "last used" value of greater than 30 minutes. Details on how to obtain these programs are provided in Appendix B, “Using the additional material” on page 221.
8.1.3 Configuring CICSPlex SM

Our CICSPlex consisted of our four listener regions (SCSCPWA1, SCSCPWA2, SCSCPWA3, and SCSCPWA4) running in our Web LPARs SC61 and SC52; and two AORs (SCSCPA1 and SCSCPAA4) running in our CICS Target LPAR SC69. Each LPAR was configured to have its own CICSPlex SM CAS (coordinating address space) and CMAS (controlling managed address space). This configuration is illustrated in Figure 76.

![Figure 76. CICSPlex configuration](image)

Each CICS listener region was connected to each AOR by means of MRO connections; the CMAS regions were connected together by CMAS-to-CMAS links. The maintenance point CMAS was SCSCPCA1 on LPAR SC61. If you are configuring or expanding your CICSPlex it is recommended that you choose a consistent naming convention, for all regions and connections, such a described in OS/390 V2R9.0-V2R10.0 Parallel Sysplex Application Migration, GC28-1863.

In addition, we set up a CICSPlex SM Web user interface (WUI) server (SCSCPA8) on LPAR SC69, in order to capture information about program usage. For further details on configuring the WUI server, refer to the CICSPlex SM Web User Interface Guide, SC34-5403.
**Workload definitions**

To manage our CICS workload, we configured CICSPlex SM in *Queue mode*. This performs workload balancing based on the length of the queues in the AORs. In order to do this, we needed to create the resources listed in Table 5.

<table>
<thead>
<tr>
<th>CICSPlex SM resource</th>
<th>CICS system group</th>
<th>WLM specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICSPlex SM command</td>
<td>CICSGRP</td>
<td>WLMSPEC</td>
</tr>
<tr>
<td>Name</td>
<td>PAA1PAA4</td>
<td>TRADER</td>
</tr>
</tbody>
</table>

Using the CICSPlex SM EUI we first created a CICS system group using the CICSGRP command; we called this PAA1PAA4 and added the AORs SCSCPAA1 and SCSCPAA4 to this group (Figure 77). We then created a WLMSPEC called TRADER, and specified the CICS system group PAA1PAA4 as the ‘Target scope’ (Figure 78). This meant that the workload controlled by this WLMSPEC was routed to the AORs in the PAA1PAA4 CICS system group (regions SCSCPAA1 and SCSCPAA4).

---

**Figure 77. CICSPlex SM — PAA1PAA4 CICSGRP**

---

**Figure 78. CICSPlex SM — TRADER WLMSPEC**

---
Next, we had to associate this WLMSPEC with our listener regions. This was done by entering the command ‘ADD’ in the CMD field alongside the WLMSPEC view entry for TRADER. This brought up the ‘Add Scope’ view as shown in Figure 79. We entered SCSCPWA1 in the ‘Scope’ field, and repeated this exercise for the other three listener regions (SCSCPWA2, SCSCPWA3, SCSCPWA4).

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TRADER routing spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spec Name</td>
<td>TRADER</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>SCSCPWA1 CICS system, group or Generic</td>
</tr>
<tr>
<td>Option</td>
<td>FORCE, NULL, or NONE for System Group</td>
</tr>
</tbody>
</table>

Press ENTER to add WLMSPEC to Scope. Type END or CANCEL to cancel without adding.

Figure 79. ‘CICSplex SM — Add Scope for WLMSPEC

Finally, we activated workload balancing in the listener regions. This was achieved by issuing the CICSSYS command, then tabbing, in turn, to the entry for each of the listener regions, and entering the command UPD in the ‘CMD’ field alongside the entry. We then paged forward until we reached the ‘Update System — Workload Attributes’ panel, and entered YES in the ‘Routing support active’ field.

To activate workload balancing in the AORs, we entered UPD in the ‘CMD’ field alongside each of the entries for the two AORs (SCSCPAA1 and SCSCPAA4) in the CICSSYS view. We then paged forward until we reached the ‘Update System - Workload Attributes’ panel, and entered YES in the ‘Target routing at startup’ field. These changes took effect as soon as the CICS regions were restarted. Our workload configuration was now complete.
8.2 Running our workload

Once we had configured our scenario, we developed a workload to test the smooth running of this solution. We used the CICS Web support version of the Trader application. This allows a Web user to buy and sell shares with one of four companies, or to query the number of shares held with those companies. For the purposes of our workload, each simulated user bought one share in a company, so each successful trade request should result in incrementing the number of shares by 1. Figure 80 illustrates our configuration.

As a reminder, the URL we used to invoke our CWS Trader application was:


Using our workload driver tool, we were able to simulate multiple instances of users all buying shares at the same time. We discovered that, if we only ran one such workload at a time, all the HTTP requests would be sent to only one of our Web LPARs (either SC61 or SC52). We ascertained that this was because the workload tool, having resolved the TCP/IP hostname in the URL, then cached that address and retained it for the entire duration of the workload. Therefore, in order to invoke full functioning of DNS connection optimization, we had to run two instances of our workload driver application simultaneously. DNS connection optimization then sent the first one to the port on one Web LPAR, and the second one to the port on the second Web LPAR. In so doing, the two separate sessions using the same TCP/IP hostname name, on the same workstation, were being resolved to two TCP/IP addresses.
Using this technique, we simulated the activity of 60 users (30 per instance of workload driver) performing together a total of 6,000 total requests. In our workload driver script, each request was a simple buy 1 activity, that is a request to purchase one share. Each buy (or sell) Web transaction results in one call to TRADERBL, and in two calls to the converter (TRADERCV) from the CWS business logic interface. Thus, for each buy 1 activity, the program TRADERCV was invoked twice, and TRADERBL once. So because we specified a total of 6,000 requests, we would expect to see a total use count of 12,000 for program TRADERCV, and a total use count of 6,000 for program TRADERBL at the end of the run. These counts would be spread across the four listener regions and the two AORs.

Lastly we also manually checked the number of shares “bought” during the workload by running a simple Trader query. This gave us the number of shares successfully traded during the workload, which should equate to the number requested in the workload driver.

When the two workloads had both successfully completed, we issued a TCP/IP “netstat” command on our Windows NT workstation, to prove that our workstation had been communicating with both LPARs. You can see from the output of that NETSTAT command in Figure 81, that we had indeed been communicating with the IP addresses of both SC61 and SC52, which is where our listener regions were configured.

![Figure 81. NETSTAT confirmation of DNS connection optimization](image)

In order to show how many times TRADERCV ran in the listener regions, and how many times the TRADERBL ran in the listener regions, we decided to capture screen images from the CICSPlex SM WUI showing usage counts for these programs. To make this as clear as possible, we first discarded the program definitions for TRADERCV and TRADERBL in all the CICS regions. This meant the first request for each program would cause a new program definition to be auto-installed, with a resulting usage count of zero. The CICSPlex SM WUI results for our workload are shown in Figure 82 and Figure 83.
These figures clearly show that as expected the 12,000 calls to the converter program TRADERCV were spread evenly by DNS connection optimization and TCP/IP port sharing evenly across the four listener regions, and the 6,000 calls to TRADERBL were spread evenly across the two AORs.
To conclude, we collected CICS statistics using the CICS statistics sample program (DFH0STAT). Full details of how to use this is given in the *CICS Performance Guide*, SC33-1699. After the workload, we then used the STAT transaction (which invokes DFH0STAT), to print out the statistics to the CICS joblog (see Figure 84). You can see that the peak number of sockets used in each listener region was fairly consistent, with either 17 or 18 sockets in use.

<table>
<thead>
<tr>
<th>Applid</th>
<th>Sysid</th>
<th>Jobname</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSCPWA1</td>
<td>PWA1</td>
<td>SCSCPWA1</td>
<td>10/17/2000</td>
<td>00:57:21</td>
</tr>
<tr>
<td>TCP/IP Service - Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Port</td>
<td>IP Address</td>
<td>Current</td>
<td>Peak</td>
</tr>
<tr>
<td>DNS.TRAD</td>
<td>8081</td>
<td>9.12.2.25</td>
<td>0</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applid</th>
<th>Sysid</th>
<th>Jobname</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSCPWA2</td>
<td>PWA2</td>
<td>SCSCPWA2</td>
<td>10/17/2000</td>
<td>00:53:17</td>
</tr>
<tr>
<td>TCP/IP Service - Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Port</td>
<td>IP Address</td>
<td>Current</td>
<td>Peak</td>
</tr>
<tr>
<td>DNS.TRAD</td>
<td>8081</td>
<td>9.12.2.25</td>
<td>0</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applid</th>
<th>Sysid</th>
<th>Jobname</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSCPWA3</td>
<td>PWA3</td>
<td>SCSCPWA3</td>
<td>10/17/2000</td>
<td>00:43:31</td>
</tr>
<tr>
<td>TCP/IP Service - Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Port</td>
<td>IP Address</td>
<td>Current</td>
<td>Peak</td>
</tr>
<tr>
<td>DNS.TRAD</td>
<td>8081</td>
<td>9.12.2.17</td>
<td>0</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applid</th>
<th>Sysid</th>
<th>Jobname</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSCPWA4</td>
<td>PWA4</td>
<td>SCSCPWA4</td>
<td>10/17/2000</td>
<td>01:01:45</td>
</tr>
<tr>
<td>TCP/IP Service - Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Port</td>
<td>IP Address</td>
<td>Current</td>
<td>Peak</td>
</tr>
<tr>
<td>DNS.TRAD</td>
<td>8081</td>
<td>9.12.2.17</td>
<td>0</td>
<td>17</td>
</tr>
</tbody>
</table>

*Figure 84. CICS statistics — TCP/IP services*

### 8.2.1 Failover scenarios

In order to really test the usefulness of our workload management configurations, we ran the following series of fail over tests. In each test we caused the outage of one particular software component, during the running of a workload consisting of 60 simulated users each invoking 1,000 buy share...
requests (a total of 6,000 requests). Following completion of the workload we analyzed the results for each component, to see where failures had occurred.

8.2.1.1 Quiesce of a listener region
In this scenario we performed a normal shutdown of the CICS listener region SCSCPWA1, during the running of the workload. The shutdown was effected using the CICSPlex SM EUI. This is illustrated in Figure 85, and the results are shown in Table 6.

![Figure 85. CWS — quiesce of a listener region](image)

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (Expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>5,993 pages complete 4 return code failures 3 early server closes</td>
<td>5,993 (6,000)</td>
<td>7</td>
</tr>
<tr>
<td>CICS listener regions</td>
<td>SCSCPWA1 - n/a SCSCPWA2 - 4,834 SCSCPWA3 - 3,016 SCSCPWA4 - 2,984</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>CICS AORs</td>
<td>SCSCPAA1 - 3,122 SCSCPAA4 - 2,871</td>
<td>5993 (6,000)</td>
<td>7</td>
</tr>
<tr>
<td>Trader</td>
<td>5,993 shares traded</td>
<td>5,993 (6,000)</td>
<td>7</td>
</tr>
</tbody>
</table>

These results show that of the 6,000 expected share trades, seven failed because of the quiesce of the listener region. On investigation we found four return code failures, were caused by four instances of the Web attach transaction (CWXN) being unable to attach the alias transaction (CWBA) during shutdown. The cause of the three early server closes was unknown.
8.2.1.2 Cancel of a listener region

In this scenario we performed a cancel of the CICS listener region SCSCPWA1, while the workload was running. This was achieved by cancelling the CICS job from SDSF. This is illustrated in Figure 86, and the results are shown in Table 7.

![Figure 86. CWS — cancel of a listener region](image)

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>5,988 pages complete 12 early server closes</td>
<td>5,988 (6,000)</td>
<td>12</td>
</tr>
<tr>
<td>CICS listener regions</td>
<td>SCSCPWA1 - n/a SCSCPWA2 - 4,958 SCSCPWA3 - 2,970 SCSCPWA4 - 3,030</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>CICS AORs</td>
<td>SCSCPAA1 - 2,823 SCSCPAA4 - 3,168</td>
<td>5,991 (6,000)</td>
<td>9</td>
</tr>
<tr>
<td>Trader</td>
<td>5,991 shares traded</td>
<td>5,9991 (6,000)</td>
<td>9</td>
</tr>
</tbody>
</table>

These results show that of the 6,000 calls to the CICS regions to trade shares, 12 calls were interrupted by the cancel of the listener region. Of these 12 calls, 3 had already completed the call to TRADERBL in the AORs, and so only 9 failures were reported. This is slightly more than in the case of the normal shutdown (quiesce), probably due to the transactions active in the listener region at the time of the cancel, which would have been abended.
8.2.1.3 Restart of a listener region
In this scenario we restarted the listener region (SCSCPWA1) while the workload was running, to see if it would be effectively brought on-line to share in the workload. This is illustrated in Figure 87, and the results are shown in Table 8.

![Figure 87. CWS — closure of a TCPIPSERVICE](image)

Table 8. CWS results — closure of a TCPIPSERVICE

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS listener regions</td>
<td>SCSCPWA1 - 1972</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA2 - 4028</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA3 - 2942</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA4 - 3,958</td>
<td>12,000 (12,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS AORs</td>
<td>SCSCPAA1 - 3,180</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>SCSCPAA4 - 2,820</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trader</td>
<td>6,000 shares traded</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
</tbody>
</table>

These results show that a CICS listener region can be re-started and brought on-line to participate in an active workload without any disruption to the success of the running workload.
8.2.1.4 Quiesce of an AOR

In this scenario we performed a quiesce of a CICS AOR (SCSCPAA1), while the workload was running. The shutdown was effected using the CICSPlex SM EUI. This is illustrated in Figure 88, and the results are shown in Table 9.

![Figure 88. CWS — quiesce of an AOR](image)

Table 9. CWS results — quiesce of an AOR

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS listener regions</td>
<td>SCSCPWA1 - 3,010</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA2 - 2,990</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA3 - 3,010</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA4 - 2,990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS AORs</td>
<td>SCSCPAA1 - n/a</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td></td>
<td>SCSCPAA4 - 4,890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trader</td>
<td>6,000, shares traded</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
</tbody>
</table>

These results show that an AOR can be successfully quiesced, without any impact on an actively running workload, when the CICSPlex SM dynamic routing program is being used to rout DPL calls into a group of AORs.
8.2.1.5 Cancel of an AOR

In this scenario we performed a cancel of a CICS AOR (SCSCPAA1), while the workload was running. This was achieved by cancelling the CICS job from SDSF. This is illustrated in Figure 89, and the results are shown in Table 10.

![Figure 89. CWS — cancel of an AOR](image)

Table 10. CWS results — cancel of an AOR

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 3,004 SCSCPWA2 - 2,996 SCSCPWA3 - 2,998 SCSCPWA4 - 3,012</td>
<td>12,000 (12,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - n/a SCSCPAA4 - 4,459</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>Trader</td>
<td>6,000, shares traded</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
</tbody>
</table>

These results show that the cancellation of an AOR is very quickly noticed by the CICSPlex SM dynamic routing program. In fact, during our workload, the cancel of the AOR had no effect on the actively running workload, probably because the LINK to the AOR from the listener regions took only a few milliseconds. We did discover that the customer VSAM RLS file needed to be defined as non-recoverable, (LOG=NONE), otherwise the cancelling of the AOR caused retained locks, which prevented any further updates to the locked record, until the failed region was restarted. This was merely done to facilitate the testing, since all the requests were for one record.
8.2.1.6 Restart of an AOR

In this scenario we restarted the AOR (SCSCPWA1) while the workload was running, to see if it would be effectively brought on-line to share in the workload. This is illustrated in Figure 90, and the results are shown in Table 11.

![Diagram of CWS - Restart of an AOR](image)

**Figure 90. CWS — restart of an AOR**

**Table 11. CWS results — restart of an AOR**

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS listener regions</td>
<td>SCSCPWA1 - 2,998</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA2 - 3,001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA3 - 2,996</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA4 - 3,004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS AORs</td>
<td>SCSCPAA1 - 645</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>SCSCPAA4 - 5,355</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trader</td>
<td>6,000, shares traded</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
</tbody>
</table>

These results show that using the CICSPlex SM dynamic routing program, a restarted AOR region can successfully take on work during an actively running workload without adversely effecting the success of any of the running transactions.
In this chapter, we demonstrate how we set up and configured a workload management solution using a Java servlet running in WebSphere Application Server Standard Edition (WebSphere AS) for OS/390. We used a servlet version of the sample Trader application. The servlet performed the presentation logic and used the CICS Transaction Gateway (CTG) to call the business logic in the CICS AORs. We used the following components to distribute work across the two LPARs in our S/390 Parallel Sysplex.

TCP/IP port sharing

TCP/IP port sharing was used to balance work across the two standalone Web servers in our Web LPAR.

EXCI

The EXCI user-replaceable module, DFHXCURM, was used to balance EXCI requests from Web servers across multiple CICS listener regions.

CICSPlex SM and dynamic routing of LINK requests

The CICSPlex SM dynamic routing program, running in Goal Mode, was used to balance LINK requests from CICS listener regions to our AORs.

VSAM Record Level Sharing (RLS)

We used the VSAM RLS feature to allow our two AORs to have shared access to the application’s VSAM files (Figure 91).

Figure 91. WebSphere OS/390 scenario
9.1 Configuring the scenario

The application we used to demonstrate workload balancing in our servlet scenario was called Trader, a sample share trading CICS application. It used a Java servlet to perform the presentation logic, and invoked the business logic in our CICS program using the External Call Interface (ECI) methods provided by the CICS Transaction Gateway for OS/390. We configured two separate LPARs in our S/390 Parallel Sysplex. On one LPAR (the Web LPAR) we configured two Web server address spaces, and balanced incoming work across these using TCP/IP port sharing. In each Web server, a WebSphere servlet engine was configured, and in this the Trader servlet ran. We did not use more than one Web LPAR because the OS/390 Web server does not support DNS connection optimization, and because of the XCF group membership limitations described in “XCF group membership” on page 84.

The ECI methods in the Java servlet were used to call the business logic in the program TRADERBL. The OS/390 CTG implements these ECI calls using the EXCI protocol when it links to the CICS listener region. We used two separate listener regions (SCSCPWA1 and SCSCPWA2) in order to remove any single points of failure. To balance requests across these regions we utilized DFHXCURM which is the EXCI User Replaceable Module (URM). This used the CICSPlex SM API to determine the available listener regions, and to balance work across them.

The listener regions then used the dynamic routing program of CICSPlex SM to balance work across the application-owning regions (AORs). The two AORs ran in a separate LPAR, and both had access to the same two VSAM files. Access to these files was shared using the facilities of VSAM Record Level Sharing (RLS). (Note that in a production environment, it would be advisable to use multiple target LPARs to provide some redundancy). This configuration is illustrated in Figure 92.

![Figure 92. Trader servlet application](image-url)
9.1.1 Configuring TCP/IP

This section describes how we utilized TCP/IP port sharing on our Web LPAR SC61 to distribute the incoming HTTP request across multiple Web servers. We only used the one LPAR because the OS/390 Web server does not support the dynamic DNS function. However, we could have used the Sysplex Distributor function to distribute IP requests to Web servers running on multiple LPARs. For further details on Sysplex Distributor, refer to 2.2.4, “Sysplex Distributor” on page 37.

To configure TCP/IP port sharing we modified the PORT definition in our TCP/IP configuration file TCPIPOE.SC61.TCPPARMS(TCPPROF), adding the SHAREPORT statement as shown in Figure 93. This allowed the two Web server jobs, WEBCICS1 and WEBCICS2 to listen on the same port 99 for incoming requests.

![Figure 93. SHAREPORT statement, for the Web servers](image)

After adding the PORT definitions, we restarted the Web servers and then used the TCP/IP command shown in Figure 94 to check that both the Web Servers were listening on the shared port 99.

![Figure 94. TCP/IP display of Web server shared port 99](image)
9.1.2 Configuring WebSphere AS and the CTG

In this section we describe how we configured WebSphere AS to deploy our CTG servlet. We used WebSphere Application Server V3.02 Standard Edition for OS/390, in conjunction with the IBM HTTP Server V5.2 and the CICS Transaction Gateway V3.1.2.

9.1.2.1 Configuring the OS/390 Web Server

We used two Web servers in standalone mode, these we named WEBCICS1 and WEBCICS2, and both ran on the Web LPAR SC61. First, we configured both Web servers to listen on the same TCP/IP port (99) for incoming requests. The details on how we did this are given in 9.1.1, "Configuring TCP/IP" on page 143. Thus, the URL http://wtsc61oe.itso.ibm.com:99 was used for invoking the Web servers.

The configuration of each Web server was identical, and we set the directive MaxActiveThreads in each Web server to 100. We also added a Hostname directive of WEBCICS1 and WEBCICS2 to the relevant server, and then configured these as aliases in our DNS server for the hostname wtsc61oe. This enabled us to determine which Web server responded to a request on port 99, since the hostname is displayed in the Web server activity statistics (Figure 124 on page 166). The JCL we used to start the Web server WEBCICS1 is shown in Figure 95.

```
//***************************************************************
//* CICS WEB SERVER PROC, HTTP V5.2, WAS V3.02
//* HFS DIR: /WEB/CICS1
//***************************************************************
//SCSWBC1 PROC P1='-B -SN SCSWBC1',
// P2='-r /web/cics1/httpd.conf',
// P3='-p 99',
// LEPARM='ENVAR("_CEE_ENVFILE=/web/cics1/httpd.envvars")'
//WEBSRV EXEC PGM=IMWHTTPD,REGION=0K,TIME=NOLIMIT,
// PARM=('&LEPARM/&P1 &P2 &P3')
//STEPLIB DD DSN=CICSTS13.CICS.SDFHEXCI,DISP=SHR
// DD DSN=CICSTS13.CPSM.SEYUAUTH,DISP=SHR
//SYSIN DD DUMMY
//OUTDSC OUTPUT DEST=* 
//SYSPRINT DD SYSSOUT=* 
//SYSERR DD SYSSOUT=* 
//STDOUT DD SYSSOUT=* 
//STDERR DD SYSSOUT=* 
//SYSSOUT DD SYSSOUT=* 
//CEEDUMP DD SYSSOUT=*
```

Figure 95. Web server startup JCL
The libraries referenced in the Web server STEPLIB concatenation (CICSTS13.CICS.SDFHEXCI and CICSTS13.CPSM.SEYUAUTH) were added so that the Web server could use the EXCI libraries (SDFHEXCI) and the CICSPlex SM API library (SEYUAUTH).

### APAR PQ49121

Notice that the -SN parameter on the PROC statement of our Web server was used as a work around to the problem reported in APAR PQ49121. This allowed us to use multiple Web servers in standalone mode on the same LPAR. Without the fix, we saw the error given below:

```
B95D860 18/Dec/2000:12:19:00.611912 : IMW0461E Error trying to obtain well known key, errno=117
```

**Note:** For details on tuning the OS/390 Web server, refer to the S/390 e-business performance Web site at:


### 9.1.2.2 Configuring WebSphere AS

On our sysplex, our WebSphere application server_root was mounted on /usr/lpp/was302/AppServer, and we created two Application server models, one for each of our Web servers. These models were mounted on separate HFS datasets mounted on the following directories: /web/cics1 and /web/cics2 (the same directories as used by our Web servers). We also used two additional HFS datasets (was/cics1 and /was/cics2) for the Websphere logs.

The Trader servlet application that we used consisted of a set of Java classes that implemented the presentation logic of the CICS Trader application, and invoked the business logic in the CICS AOR using an ECI call. The root servlet was called traderb2.class. In order to deploy this Trader servlet, we added the following properties shown in Figure 96, to our Application Server configuration file (was.conf). For details on how to obtain this sample Trader servlet code, refer to Appendix A, “The Trader application” on page 205.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployedwebapp.cics.host</td>
<td>default_host</td>
</tr>
<tr>
<td>deployedwebapp.cics.rooturi</td>
<td>/servlet</td>
</tr>
<tr>
<td>deployedwebapp.cics.classpath</td>
<td>/web/cics1/servlets</td>
</tr>
<tr>
<td>webapp.cics.servlet.traderServlet.code</td>
<td>traderb2</td>
</tr>
<tr>
<td>webapp.cics.servlet.traderServlet.servletmapping</td>
<td>/traderb2</td>
</tr>
</tbody>
</table>

*Figure 96. Configuration of OS/390 was.conf file*
The meaning of the statements shown in Figure 96 is as follows:

1. Specifies the virtual host in WebSphere.
2. Specifies the path by which the servlet is known to the local host.
3. Specifies the classpath that is used by the application level class loader. The application level class loader tries to locate the servlet if the system class loader cannot find the servlet in the WebSphere classpath.
4. Specifies the class file of the Trader servlet to its Web application. “code=traderb2” indicates that the servlet has been compiled in file traderb2.class. Trader.Servlet is the unique name of the servlet.
5. Specifies that you have to enter “traderb2” after the root URL in order to invoke the Trader servlet. Therefore, we used the URL http://wtsc61oe.itso.ibm.com:99/servlet/traderb2 to invoke the servlet.

For further details on configuring WebSphere AS on OS/390, refer to HTTP Server Planning, Installing, and Using, SC31-8690.

9.1.2.3 Configuring the CTG

Since the servlet was running within the OS/390 Web server and connecting to CICS regions on the OS/390 LPAR, we were able to utilize the CTG local: protocol, meaning that we did not need to use the CTG Java gateway application, but instead the CTG ECI Java methods executed within the Web server address space. The CTG then utilized the EXCI protocol to send the ECI requests into the CICS region. The flow of control in our Trader servlet application is illustrated in Figure 97. For further details on how the OS/390 CTG function, refer to 1.2.2, “CICS Transaction Gateway” on page 14.

Figure 97. OS/390 CTG Trader application
Adding the CTG libraries to WebSphere

We modified the Java classpath used by the WebSphere Application Servers to include the CTG jar files, and the libpath to include the /usr/lpp/ctg/bin directory. The classpath is used to locate the compiled Java classes, the libpath is used by the application server to load the CTG native library libCTGJNI.so which is the interface between the Java code and the EXCI. The modifications were made to both configuration files (/web/cics1/was.conf and /web/cics2/was.conf), and are illustrated in Figure 98.

```java
appserver.classpath=/usr/lpp/ctg/classes/ctgclient.jar:/usr/lpp/ctg/classes/ctgserver.jar
appserver.libpath=/usr/lpp/ctg/bin
```

Figure 98. WebSphere classpath and libpath for OS/390

DFHXCLOPT table

Finally, when using the CTG, you will probably need to modify your DFHXCLOPT table. This is the table that controls the EXCI options. In particular, it controls the surrogate userid checking using the parameter SURROGCHK. This parameter defaults to YES, meaning that if you flow a userid from the CTG on the ECI call, you will need to define a RACF surrogate profile to allow the userid of the CTG address space to switch to this userid. You do this by defining READ authority to a profile named userid.DFHEXCI in the SURROGAT general resource class.

We did not have CICS security active in our tests and so did not flow userids on our ECI requests, thus the userid and password strings in the ECIRequest object defaulted to null. This has the effect that the userid of the job (in our case, the Web server) was flowed as the userid in the EXCI DPL_Request created from the ECI call. Since this userid is already the userid of the address space that the CTG address job runs under, surrogate userid checking is not required, and so we did not need to alter the DFHXCLOPT parameter of SURROGCHK from its default of YES. However, if you were to specify a userid on the ECI call, you would need to either add the correct RACF profile, or modify the SURROGCHK parameter.

Configuring the EXCI pipes

Since the CTG uses the EXCI to send ECI requests to the listener CICS regions, we needed to configure an EXCI connection from each Web server to each CICS listener region. EXCI connections use the concept of a pipe; this is a communication path between a sending process (the CTG) and the receiving CICS region. Each pipe maps onto one CICS MRO session. We chose to use specific named pipes, so that each Web server used a different connection to each CICS region; this aided us in our analysis of session
usage for each Web server. It is also possible to use generic pipes, when you
only have one CICS listener region. This allows multiple client programs to
share the same connection into CICS. For more information on the EXCI
interface, refer to CICS External Interfaces Guide, SC33-1944.

To configure which connection was used by which Web server, we set the
CTG variable DFHJVPIPE to the relevant netname for the required
connection. This was configured in each Web server’s environment variables
file (httpd.envvars) as shown below:

```
DFHJVPIPE=CTGPIPE1
```

We used the netname CTGPIPE1 for the Web server WEBCICS1, and the
netname CTGPIPE2 for the Web server WEBCICS2. For details on how we
configured the matching MRO connections in our CICS listener regions, see
9.1.3, “Configuring CICS” on page 150.

**DFHXCURM**
The EXCI user-replaceable module (DFHXCURM) receives control before
every Allocate_Pipe call, and also after an EXCI command fails with a
retryable response. (For more details on retryable responses, refer to 5.1.3,
“EXCI” on page 82). Its aim is to allow simple workload balancing by providing
a means to modify the APPLID specified in the Allocate_Pipe call. Note that
the DFHXCURM itself cannot retry a failed command, it can only store the
information about the previous retryable errors.

During this project, we developed a version of DFHXCURM that did workload
balancing for EXCI calls across our group of listener regions in our CICSPlex,
in order to provide for failover and workload balancing. Our logic used the
CICSPlex SM API to obtain the status of the listener regions before every
Allocate_Pipe, by performing the command:

```
EXEC CPSM GET OBJECT(MCICSRGN) CRITERIA(SCSCPWA*)
```

This logic is illustrated in Figure 99. To obtain our sample DFHXCURM code,
refer to Appendix B, “Using the additional material” on page 221.

```plaintext
If ALLOCATE then
  EXEC CPSM GET OBJECT(MCICSRGN) CRITERIA(SCSCPWA*) to
  obtain list of active CICS regions, and number of tasks in
  each region
  Set APPLID by selecting region with fewest tasks

If Retryable response then exit
```

*Figure 99. Sample DFHXCURM logic*
We found in our initial tests that this design had a measurable overhead when used with the CTG V3.1.1 due to the high number of Allocate_Pipe calls issued, and thus the high number of calls to CICSPlex SM. However, when used with the CTG V3.1.2, no measurable overhead was detected, since only one Allocate_Pipe call is ever issued per thread (TCB).

As an alternative to our DFHXCURM design using CICSPlex SM, we also considered a design that stores region status within the URM, similar to the example logic provided with DFHXCURM. This region status can be gathered from the responses to EXCI retryable errors and so would not require use of CICSPlex SM. However, such a design would have been harder to implement for the following reasons:

- The sample DFHXCURM logic stores region status in global storage that is private to each TCB (remember an ECI request runs in its own Web server thread or TCB). Thus to work efficiently, additional logic would be required to allow sharing of storage area across different TCBs, and would also require corresponding synchronization logic to prevent multiple threads updating such shared storage at the same time.

- Regions marked as inactive would need to be retried after a given interval. This could be after a certain time has elapsed, or after a certain number of EXCI calls. If using the CTGV3.1.2 the number of calls to DFHXCURM will be very low, so elapsed time would have to be used.

- Although such a design does have the advantage that it could handle the EXCI NO_PIPE retryable response, this situation if it occurs, is most likely to be because the limitation of 100 pipes per EXCI address space has been reached, and not the maximum number of CICS sessions has been (since this can be set up to 999). Since this situation cannot be remedied by the application itself there is little value in handling it.

For further details on programming with the EXCI, refer to Chapter 9, “The EXCI CALL Interface”, in the CICS External Interfaces Guide, SC33-1944.
9.1.3 Configuring CICS

This section describes how we configured our listener regions and AORs for our OS/390 CTG Trader servlet scenario (see Table 12).

Table 12. CICS resources used by the Trader servlet application

<table>
<thead>
<tr>
<th>Listener Regions</th>
<th>Transaction</th>
<th>Program</th>
<th>File</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TRB2</td>
<td>TRADERB2</td>
<td></td>
<td>CTG1, CTG2</td>
</tr>
<tr>
<td>AORs</td>
<td>TRB2</td>
<td>TRADERBL</td>
<td>CUSTFILE</td>
<td>COMPFILE</td>
</tr>
</tbody>
</table>

Program definitions

We used the CICSPlex SM dynamic routing program to route the requests from our listener regions (SCSCPWA1, SCSCPWA2) to our AORs (SCSCPAA1, SCSCPAA4). To enable this we added the SIT parameter DTRPGM=EYU9XLOP to all our listener regions, which specifies that the CICSPlex SM supplied dynamic routing program is to be used.

Then we created a definition for the program TRADERB2. This was defined in the listener regions with the DYNAMIC option set to YES to enable dynamic program routing to control the AOR to which the program was routed. In the AORs we did not create a program definition for the program TRADERBL, but allowed program autoinstall to create one instead. The definition of program TRADERB2 is illustrated in Figure 100. Note that in the definition of program TRADERB2 we specified a TRANSID of TRB2, as our private mirror transaction.

![Figure 100. Program definition for TRADERB2](image-url)
Having defined the program, the last thing we had to do was to enable data conversion for the data passed in the COMMAREA from program TRADERB2 to our servlet. To achieve this we added a TYPE=SELECT macro to our CICS DFHCNV data conversion table, with a length of 372 bytes (which is the length of the COMMAREA used by TRADERB2.) Note that we had to specify an ASCII client code page (CLINTCP=437), since the trader servlet expects to receive data the COMMAREA data as an ASCII byte, even when running on OS/390. This is to ensure cross platform compatibility. Our DFHCNV entry is shown in Figure 101.

```
DFHCNV TYPE=ENTRY, RTYPE=PC, RNAME=TRADERB2, CLINTCP=437
   DFHCNV TYPE=SELECT, OPTION=DEFAULT
   DFHCNV TYPE=FIELD, OFFSET=0, DATATYP=CHARACTER, DATALEN=372,
                     LAST=YES
```

Figure 101. DHFCNV entry for program TRADERB2

We then reassembled and link edited the DFHCNV table, and reloaded the table into our CICS listener regions. For further details on DFHCNV conversion templates, refer to the manual Communicating from CICS on System/390, SC33-1697.

Transaction definition

Because we configured CICSPlex SM to use the MVS WLM in “goal mode”. We had to define a private mirror transaction in the AORs for the dynamic routing program. We decided on the name TRB2 for our private mirror transaction. We installed it in the listener regions and the AORs. For more details on configuring MVS WLM in this scenario, refer to “MVS Workload Manager definitions” on page 157.

Specifying a private mirror transaction is also recommended because it enables you to easily analyze program usage for a given application, without including requests to CPMI.
The definition of our TRB2 private mirror transaction is shown in Figure 102. It was created by copying the supplied CPMI mirror transaction.

![Figure 102. Transaction definition for mirror TRB2](image)

**File definitions**

The VSAM file definitions for the files COMPFILE and CUSTFILE were defined in RLS mode to enable the files to be shared between the two AORs, SCSCPAA1 and SCSCPAA4. The COMPFILE is used by Trader to store a static list of company data, and the CUSTFILE is used to store the number of shares held.

For further details on creating the CICS definitions for these files refer to see 8.1.2, “Configuring CICS” on page 121 for more details.
EXCI definitions
CICS requires a connection and sessions definition pair for each EXCI connection. We defined a connection/sessions pair name CTG1 and EXCGSESS for the Web server WEBCICS1 (as shown in Figure 103 and Figure 104), and another pair named CTG2 and EXCGSESS for the Web server WEBCICS2. The netname of CTGPIPE1 matched the name of the EXCI pipe used by the CTG in WEBCICS1, and specified in the DFHJVPIPE environment variable. Note also that we specified a receive count of 200 sessions; this is the maximum number of sessions that could theoretically be used from both Web servers to any one CICS region.

![Figure 103. EXCI connection definition CTG2](image)

![Figure 104. EXCI sessions definition EXCGSESS](image)
9.1.4 Configuring CICSPlex SM

In this section we explain how we configured CICSPlex SM in conjunction with MVS Work Load Manager (WLM) to actively manage the routing of our Trader workload. We used WLM in goal mode as opposed to queue mode, which we used in our previous scenario (see Chapter 8, “CICS Web support” on page 111). Goal mode is a more advanced feature of MVS WLM which allows you to define a specific response time goal that you wish to be met for the desired transactions, and WLM will strive to achieve this goal in all your target CICS regions. Using goal mode requires more definitions to be made in WLM, and may be worthwhile if you have specific response time targets in your service level agreements.

In this scenario our CICSPlex was spread across two LPARs. We configured each LPAR to have its own CAS (coordinating address space) and CMAS (controlling managed address space). These CMAS regions were connected together by CMAS-to-CMAS links. The maintenance point CMAS was SCSCPCA1 on LPAR SC61. In addition, we set up a CICSPlex SM Web User Interface (WUI) server (SCSCPAA8) on LPAR SC69 which we used for CICSPlex SM operational procedures, such as capturing information about program usage. This is illustrated in Figure 105.

Figure 105. CICSPlex configuration
CICSPlex SM definitions

We used the CICSPlex SM workload manager to control the dynamic routing of the Trader application using, in conjunction with MVS WLM goal mode. In order to do this we needed modify or create the following CICSPlex SM resources, as shown in Table 13.

Table 13. CICSPlex SM resources — Goal mode

<table>
<thead>
<tr>
<th>CICSPlex SM resource</th>
<th>CICS system group</th>
<th>WLM specification</th>
<th>Tran group</th>
<th>Workload Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICSPlex SM command</td>
<td>CICSGRP</td>
<td>WLMSPEC</td>
<td>TRANGRP</td>
<td>WLMDEF</td>
</tr>
<tr>
<td>Name</td>
<td>PAA1PAA4</td>
<td>TRADER</td>
<td>TRB2TRAN</td>
<td>TRB2</td>
</tr>
</tbody>
</table>

The first step in configuring CICSPlex SM for our workload was to UPDATE our WLMSPEC. To achieve this, we used the same CICSPlex SM End User Interface (EUI) to modify our TRADER WLMSPEC that we previously used in Chapter 8, “CICS Web support” on page 111. We specified that we wanted to invoke the ‘GOAL’ algorithm, and thus use MVS WLM in goal mode, as opposed to queue mode, which we used previously (see Figure 106).

---

```
COMMAND ->

WLMSPEC Name TRADER
Description >>> TRADER Routing Spec
Affinity Relation -> Default Affinity Relation
                    (USERID, LUNAME, GLOBAL, RAPPL)
Affinity Lifetime  -> Default Affinity Lifetime
                    (SIGNON, LOGIN, SYSTEM, PERMANENT, PCONN, DELIMIT,
                    ACTIVITY, PROCESS)
Host Key            -> USERID  Default Primary search criteria
                    (USERID, LUNAME)
Create Affinity     -> N/A     Create auto affinity (YES, NO, N/A)
Target Scope        -> PAA1PAA4 Default CICS System, Group or Generic
Event Name          -> TRADER, STATDEF or Generic
Abend Health        -> 0      Target ABEND Health Factor (0 - 99)
Abend Load          -> 0      Target ABEND Load Factor (0 - 99)
Algorithm Type      -> GOAL   Algorithm Type (GOAL, QUEUE)
```

Press ENTER to update the WLMSPEC. Type END or CANCEL to cancel without updating.

---

Figure 106. CICSPlex SM — TRADER WLMSPEC
Next we create a TRANGRP called TRB2TRAN (Figure 107) and added our private mirror transaction TRB2 to this group.

--- Create Trangroup for SCSIPLEX ---

**COMMAND**

- **Trangroup name**: TRB2TRAN
- **Description**: Transgrp to dyn. route TRADERBL
- **Status**: ACTIVE
- **Match Key**: USERID
- **Affinity Relation**: Optional affinity relation may be:
- **Affinity Lifetime**: Optional affinity lifetime may be:
- **Create Affinity**: N/A
- **Event Name**: MDEFTDMDEFT or generic
- **Abend Health**: 0
- **Abend Load**: 0

Press ENTER to create trangroup.
Type END or CANCEL to cancel without creating.

--- Create Workload Definition for SCSIPLEX ---

**COMMAND**

- **Definition Name**: TRB2
- **Description**: Workload Def for TRB2
- **Trangroup Name**: TRB2TRAN
- **Terminal Luname**: *
- **User ID**: *
- **Process Type**: Specific Process Type or *
- **Target Scope**: PMAPNA

Press ENTER to create definition.
Type END or CANCEL to cancel without creating.

Figure 107. CICSPlex SM — Create TRANGRP

We then created a WLMDEF called TRB2. This was done by entering the command ‘ADD’ in the CMD field alongside a WLMDEF view, this brought up the ‘Create Workload Definition’ view as shown in Figure 108.

Figure 108. CICSPlex SM — WLMDEF TRB2
We activated this workload by inserting the WLMDEF TRB2, into the WLMSPEC (TRADER). This was achieved using the command ‘INS’ in the CMD field alongside the WLMDEF view for TRB2 (Figure 109).

Figure 109. CICSPlex SM — Install WLMDEF

**MVS Workload Manager definitions**

In this section we explain how we set up a service class in MVS Workload Manager to which we associated the CICS transaction TRB2, which is the mirror program TRADERB2 was routed under. The name of the service class was TRAD which had an average response time of 0.02 seconds.

First of all we entered WLM from the TSO command line, and received the initial WLM screen, shown in Figure 110.

Figure 110. MVS WLM — Initial welcome screen
We then pressed Enter and received the panel in Figure 111.

**Figure 111. MVS WLM — choose service definition**

We selected option 2 to get the Definition Menu, as shown in Figure 112.

**Figure 112. MVS WLM — definition menu**

We entered 4 on the Definition Menu to get a list of existing service classes as shown in Figure 113 on page 159.
We entered action code 1 to create a new service class, and were then presented with the Create a Service Class panel (Figure 114).

In this panel we specified the name of the service class we wanted to create (TRAD) and in which existing MVS workload we wanted to include the new class. We chose service class name TRAD and the MVS workload called CICS. When we entered an i in the action field, we were presented with a pop-up screen, as shown in Figure 115.
We chose option 1, ‘Average response time’ from this panel, which presented us with the panel in which we specified the response time goal of 0.02 seconds (Figure 116).
We were then presented with the updated input screen shown in Figure 117.

![Figure 117. MVS WLM — response times defined](image)

We pressed PF3 from this screen to save the newly defined service class and received confirmation that it had been created, as shown in Figure 118.

![Figure 118. MVS WLM — service class TRAD created](image)
Next, having returned to the main menu, we then took the necessary steps to associate transaction TRB2 with this service class. We selected option 6, 'Classification Rules' as shown in Figure 119. We were then presented with the Subsystem Type Selection List for Rules panel (Figure 120).

Figure 119. MVS WLM — add TRB2 to service class TRAD (1)

Figure 120. MVS WLM — add TRB2 to service class TRAD (2)
In this panel, there was a subsystem entry called CICS, and it was in this entry that we wanted to add TRB2. We selected option 3, which allowed us to modify the entry, and presented us with the panel shown in Figure 121.

![Figure 121. MVS WLM — add TRB2 to service class TRAD (3)](image)

This panel showed all the current rules for the subsystem type CICS. To add an entry for TRB2, we entered an I in the action column, which allowed us to enter our new data (Figure 122). We inserted an entry for transaction TRB2 and associated it with service class TRAD. Pressing PF3 saved our MVS WLM definition.

![Figure 122. MVS WLM — add TRB2 to service class TRAD (4)](image)
9.2 Running our workload

Once we had configured our scenario we developed a workload to test the smooth running of the Trader servlet solution on WebSphere OS/390. Figure 123 illustrates our configuration.

Using our workload driver, we were able to simulate 60 instances of Trader users all trading at once. For the purposes of our workload, each user performed one buy share activity and bought just one share in a company. Thus for each successful trade request, the number of shares (as held in the customer VSAM file) would increment by one. We specified a total of 6,000 total requests. For each buy share activity, the Trader servlet is coded to make two calls to the CICS business logic, one to buy the share and one to get a quote of the updated share values.

So, for our 6,000 total requests, we expected to see a total of 12,000 calls to TRADERB2 distributed across the two listener regions. These calls would then by dynamically routed to the program TRADERBL in the AORs. We also expected an even distribution of requests across the two Web servers due to the port sharing we implemented.

We used the CICSPlex SM Web User Interface (WUI) to provide us with information about CICS program usage in the listener regions and AORs, together with the OS/390 Web server activity statistics to provide us with activity statistics about the Web servers.
DFHXCURM

Our DFHXCURM was coded to provide simple workload balancing of EXCI requests across our two listener regions. This was achieved by using the CICSPlex SM API within the exit to discover the active CICS regions before each EXCI Allocate_Pipe call was made.

However, we did find that when running our workloads, using the V3.11 of the CTG, that the overhead of issuing the EXEC CPSM GET OBJECT(MCICSRGN) call for every EXCI Allocate_Pipe, caused our workload to run at about 75% of the throughput it would otherwise have done. However, we subsequently found that once we upgraded to V3.12 of the CTG, there was no noticeable degradation in performance, due to the EXCI pipe reuse modification in this release. For further details on the CTG pipe reuse modification, refer to 5.1.3, “EXCI” on page 82.

One problem we also encountered in developing our custom DFHXCURM was that our trace messages from the module (assembler WTO statements), caused D23 abends in the Web server when used under load. We recommend that you should only use such WTO statements when debugging problems in DFHXCURM, and only when running requests in series.

Web Servers

The OS/390 Web server provides a useful utility to monitor server activity. To access this utility, we performed the following steps:

2. First we clicked on the link Server Administration.
3. Next we clicked on Server Activity Monitor in the left hand panel.
4. In the Server Activity Monitor section we clicked Activity Statistics.

Since both our Web servers were listening on the same port, we were able to obtain the statistics by clicking on the Refresh button, then stopping the Web server that had just responded, then clicking the Refresh button again so that the second Web server responded to the request. The statistics for our Web servers WEBCICS1 and WEBCICS2 are shown in Figure 124 and Figure 125. We can see that Web server WEBCICS1 processed 2970 request and the other Web server WEBCICS2 processed 3030 requests, a total of 6,000 requests as expected.
Figure 124. Activity statistics for Web server WECICS1

Figure 125. Activity statistics for Web server WECICS2

166 Workload Management for Web Access to CICS
CICS program usage

Finally, we turned to the CICSPlex SM WUI to check the number of times that the TRADERB2 program had been invoked on the listener regions, and the number of times that the TRADERBL program had been invoked on the two AORs. The screen capture of the WUI statistics for the usage of program TRADERBL in the AORs is shown in Figure 126.

![Figure 126. CICSPlex SM WUI display — usage count for TRADERBL](image)

You can see that CICS region SCSCPA1 processed 5320 requests and region SCSCPA4 processed 6680 a total of 12,000 requests as expected.

To measure the usage of TRADERB2 in the listener regions we had to monitor the usage of the data conversion table DFHCNV. This is because program TRADERB2 is only a dynamic definition, and so does not actually run, but is dynamically routed to the AORs, but still invokes the data conversion routines in the listener regions. The statistics showed that 6,531 requests were made to region SCSCPWA1 and 5,469 requests made to region SCSCPWA2, again a total of 12,000 requests.
9.2.1 Failover scenarios

In order to really test the usefulness of our workload management configurations, we ran the following series of failover tests. In each test we caused the outage of one particular component, during the running of a workload consisting of 60 simulated users, each invoking 1,000 buy share requests (a total of 6,000 requests). Following completion of the workload, we analyzed the results for each component, to see where failures occurred.

9.2.1.1 Cancel of a Web server

In this scenario we terminated the OS/390 Web server, by cancelling the job from SDSF during the running of the workload. This is illustrated in Figure 127, and the results are shown in Table 14.

![Figure 127. WebSphere OS/390 — cancel of a Web server](image)

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (Expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>5,983 pages complete 32 early closes</td>
<td>5,983 (6,000)</td>
<td>17</td>
</tr>
<tr>
<td>Web servers</td>
<td>WEBCICS1 - n/a</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td></td>
<td>WEBCICS2 - 5,002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 9,666 a</td>
<td>11,951 (12,000)</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>SCSCPWA2 - 2,285 a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - 5,502</td>
<td>11,940 ((12,000)</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>SCSCPAA4 - 6,449</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trader</td>
<td>5,983 shares traded</td>
<td>5,967 (6,000)</td>
<td>17</td>
</tr>
</tbody>
</table>

These results indicate the ability of TCP/IP port sharing to detect a failed Web server and to rapidly send the IP traffic to the other Web server listening on the same port. The failures in this situation are attributable to the number of servlet transactions active in any one Web server at a given moment.
9.2.1.2 Quiesce of a listener region

In this scenario we performed a normal shutdown of the CICS listener region SCSCPWA2, during the running of the workload. The shutdown was achieved using the CICSPlex SM EUI. This is illustrated in Table 128, the results are shown in Table 16.

![Diagram showing quiesce of a listener region](image)

Table 15. WebSphere OS/390 — quiesce of a listener region

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (Expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>Web servers</td>
<td>WEBCICS1 - 2,968</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>WEBCICS2 - 3,032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS listener regions</td>
<td>SCSCPWA1 - 8,044</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>(DFHCNV calls)</td>
<td>SCSCPWA2 - n/a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS AORs</td>
<td>SCSCPAA1 - 1,620</td>
<td>11,313 (12,000)</td>
<td>687</td>
</tr>
<tr>
<td>(TRADERBL calls)</td>
<td>SCSCPAA4 - 9,693</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trader</td>
<td>5,645 shares traded</td>
<td>5,645 (6,000)</td>
<td>355</td>
</tr>
</tbody>
</table>

These results show that of the 12,000 expected call to TRADERBL, 687 calls failed before they reached the CICS AORs and thus 355 of the calls to buy shares were not processed. The reason for this large number of failed calls is not known, but may be due to the fact that our DFHXCURM did not obtain information about the outage quickly enough when using the CPSM monitoring view.
9.2.1.3 Cancel of a listener region

In this scenario we performed a cancel of the CICS listener region SCSCPWA2, while the workload was running. This was achieved by cancelling the CICS job from SDSF. This is illustrated in Figure 129, and the results are shown in Table 17.

![Figure 129. WebSphere OS/390 — cancel of a listener region](image)

Table 16. WebSphere OS/390 results — cancel of a listener region

<table>
<thead>
<tr>
<th>Component</th>
<th>ResultS</th>
<th>Successes (Expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>Web server</td>
<td>WEBCICS1 - 2,975, WEBCICS2 - 3,025</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 6,832, SCSCPWA2 - n/a</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - 3,918, SCSCPAA4 - 8,049</td>
<td>11,967 (12,000)</td>
<td>33</td>
</tr>
<tr>
<td>Trader</td>
<td>5,992 shares traded</td>
<td>5,968(6,000)</td>
<td>32</td>
</tr>
</tbody>
</table>

These results show that of the 12,000 expected call to TRADERBL application in the AORs, 33 calls failed before they reached the CICS AORs, and thus 32 of the calls to buy shares were not processed.

We note that in this case there were a lot less failures than the normal shutdown (see 9.2.1.2, “Quiesce of a listener region” on page 169) we speculate this is because the cancel of the region terminated the pipes quicker than the normal shutdown.
9.2.1.4 Restart of a listener region

In this scenario we started the workload with just one listener region (SCSCPWA1) active, and then started the second listener region SCSCPWA2 while the workload was running. This is illustrated in Figure 130, and the results are shown in Table 17.

![Figure 130. WebSphere OS/390 — restart of a listener region](image)

<table>
<thead>
<tr>
<th>Component</th>
<th>ResultS</th>
<th>Successes (Expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>Web server</td>
<td>WEBCICS1 - 2,974 WEBCICS2 - 3,026</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 12,003 SCSCPWA2 - 0</td>
<td>12,003 (12,000)</td>
<td>None</td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - 5,224 SCSCPAA4 - 7,442</td>
<td>12,000 (12,000)</td>
<td>None</td>
</tr>
<tr>
<td>Trader</td>
<td>6,000 shares traded</td>
<td>6,000 (6,000)</td>
<td>None</td>
</tr>
</tbody>
</table>

These results show that a restarted CICS listener region is not brought on-line during the running of a normal workload. The reason for this is the EXCI pipe usage in V3.12 of the CTG, whereby EXCI pipes stay allocated until either a retryable error occurs or the MRO connection is closed. The extra three calls made to DFHCNV in the listener regions are because DFHCNV is invoked three times when a CICS region is started.

To test this theory, we restarted the Web servers and ran the test again. This time the workload was correctly shared across both CICS listener regions, since the termination of the Web server address spaces closed the EXCI connections. These results show that workload balancing using DFHXCURM can only provide failover, and not workload balancing. For further details on EXCI pipe usage, refer to “Configuring the EXCI pipes” on page 147.
Chapter 10. CTG servlets with WebSphere AS (Windows NT)

In this chapter, we demonstrate how we set up and configured a workload management solution using a Java servlet running in WebSphere Application Server Standard Edition (WebSphere AS) for Windows NT, invoking a CICS program using the External Interface Calls (ECI) via the CICS Transaction Gateway (CTG). We used the following components to distribute work across our Windows NT machines, OS/390 LPARs and CICS regions. Figure 131 illustrates this scenario.

**Network Dispatcher**

We used Network Dispatcher to balance work across two Windows NT machines running WebSphere AS.

**CICS Universal Client Workload Manager**

The CICS Universal Client Workload Manager allowed us to distribute the ECI requests across all four of our CICS listener regions.

**CICSPlex SM and dynamic routing of LINK requests**

The CICSPlex SM dynamic routing program, function in Goal Mode, was used to balance LINK requests from our CICS listener regions to our AORs.

**VSAM Record Level Sharing (RLS)**

We used the VSAM RLS feature to allow our two AORs to have shared access to the application’s VSAM files.

---

Figure 131. WebSphere and CTG on Windows NT scenario
10.1 Configuring the scenario

The application we used to demonstrate workload balancing in our servlet scenario was called Trader, a sample share trading CICS application. It used a Java servlet to perform the presentation logic, and invoked the business logic in our CICS program using the ECI methods provided by the CICS Transaction Gateway for Windows NT.

The ECI methods in the Java servlet were used to call the business logic in the CICS program TRADERBL. The Windows NT CTG implements these ECI calls using the integrated CICS Universal Client, which uses the LU6.2 protocol to send requests to CICS.

We used four separate CICS listener regions, two in LPAR SC61 and two in another LPAR SC52. This allowed us to distribute the incoming work and also to remove any single points of failure.

The router regions then used the dynamic routing program of CICSPlex SM to balance work across the CICS application-owning regions (AORs). The two AORs ran in a separate LPAR, and both had access to the same two VSAM files. Access to these files was shared using the facilities of VSAM Record Level Sharing (RLS). This configuration is illustrated in Figure 132. Note that in a production environment, it would advisable to use multiple target LPARs to provide some redundancy, and you would most likely be using a much larger number of CICS AORs.

![Figure 132. Trader servlet application](image)

In the following sections we discuss the configuration of Network Dispatcher, WebSphere AS, and the CTG. For further information on configuring CICS or CICSPlex SM, refer to 9.1.3, “Configuring CICS” on page 150 in Chapter 9, “CTG servlets with WebSphere AS (OS/390)”. 

174 Workload Management for Web Access to CICS
10.1.1 Configuring Network Dispatcher

Network Dispatcher enables TCP/IP packet distribution to a cluster of target machines. Our forwarding cluster address, nassau, was located on our dispatcher machine bromine, and forwarded all requests for nassau straight onto our WebSphere machines loire and malawi. This forwarding cluster address is an additional IP address used solely for Network Dispatcher workload balancing. It must not be used by any other computer on the network. The WebSphere machines also had a loopback adaptor installed to enable them to accept packets destined for nassau. All machines were in the TCP/IP domain almaden.ibm.com. Figure 133 illustrates our configuration.

![Network Dispatcher topology](image)

**Figure 133. The Network Dispatcher topology**

**The Network Dispatcher configuration wizard**

First, we installed Network Dispatcher V3 (WebSphere Edge Server V1.0) on bromine. We then started the configuration wizard and performed the following steps, starting by defining the forwarding cluster address. We specified the host nassau (9.1.150.231) when prompted (Figure 134).

![Defining a Cluster](image)

**Figure 134. Network Dispatcher — defining a cluster**
We then needed to specify a port to monitor for request forwarding. All HTTP packets reaching the Network Dispatcher machine on this port, destined for the forwarding cluster address, will be forwarded to our target machines. We specified port 80, on which our Web servers would listen (see Figure 135).

Next we specified the two servers, loire and malawi, to do workload balancing across. These machines can be seen listed in the server list in Figure 136.
Aside from just monitoring the number of network connections, Network Dispatcher is also capable of gathering feedback information about target servers. To perform this function, advisors are used. We started the supplied HTTP advisor, which will continually measure the response time of HTTP packets sent to port 80 on the target machines. Note that a better alternative could have been, to write a WebSphere customer servlet advisor that called CICS via the CTG. For more details on this option, refer to “Advisors” on page 87.

**Configuring the NT loopback adaptor**

A loopback adaptor is required to enable packets, originally destined for the workload balancing cluster address (nassau), to be accepted by the cluster WebSphere host machines, loire and malawi. The Network Dispatcher documentation gives instructions on how to configure the required loopback adapter, but we give a brief outline of the steps we took.

On Windows NT you need to install the MS loopback adaptor from the Windows NT installation CD. Upon installation you are then prompted to choose the frame type, and to configure the IP address of the loopback adapter. We chose the default frame type and entered the IP address of the workload balancing cluster (9.1.250.231). The WebSphere AS host machine was then capable of accepting IP packets destined for the workload balancing cluster. Figure 137 illustrates the Windows NT loopback configurations screens.

![Figure 137. Configuring the MS loopback adaptor](image)
An effect of adding the MS loopback adapter is also to modify the TCP/IP routing table for the machine, adding an entry for the new address, that is not required.

To inspect the routing table and remove the duplicate entry, we used the TCP/IP route command from the Windows NT command prompt, as shown in Figure 138. This had to be performed on both the server machines (bromine and loire).

First, we entered the command `route print`, which shows the current IP interfaces and the active routes. We then deleted the extra route using the command `route delete 9.0.0.0`, since we already have a route to the cluster address (9.1.250.231) lower down in the table. More details on inspecting the routing tables can be found in the Network Dispatcher Users Guide, GC31-8496.

![Figure 138. Network Dispatcher — inspecting the routing table](image)
Connecting to the dispatcher

Our configuration was now complete. We selected to connect to the dispatcher host and were presented with a graphical view of configuration, shown in Figure 139.

![Figure 139. Connecting to the Dispatcher host machine](image)

Our host bromine has an Executor in which the cluster address 9.1.250.231 is configured. We have also started the HTTP advisor to monitor port 80 usage on our two servers 9.1.150.48 and 9.1.150.43. We did not configure or use the Interactive Session Support of Content Based Routing features.
Changing the weights

Finally, we modified the attributes of the HTTP Advisor within the Manager to ensure a smooth response to our workload. We changed the **Proportion given to active connections**, **Proportion given to new connections** and the **Proportion given to the port** to 40, 30 and 30 respectively. We also increased the **smoothing index** to 10 (Figure 140).

We based these alterations on experimentation during the initial running of several of our workloads. We found that before we made these changes, the number of requests forwarded to our server machines was likely to “sawtooth”, where one machine is initially heavily loaded, and the response time degrades, and so the other machine then becomes heavily loaded, and so on.
10.1.2 Configuring WebSphere AS and the CTG

In this section we describe how we configured WebSphere Application Server to deploy our servlet that used CICS Transaction Gateway (CTG) to invoke the Trader application in our CICS regions. We used WebSphere Application Server Standard Edition for Windows NT V3.02 (with fixpack 1), together with the IBM HTTP Server V1.3.6.2 for Windows NT and the CICS Transaction Gateway V3.11 (which includes the CICS Universal Client).

Configuring the IBM HTTP Server

We installed and configured two Web servers: one on loire, and one on malawi. No special manual configuration is required to use the IBM HTTP Server with WebSphere AS, since the WebSphere installation process performs this step.

Configuring the CTG

Since our servlet was to run on the same machine that we had configured the CICS Universal Client, we were able to utilize the CTG local: protocol, meaning that we did not need to use the CTG Java gateway application, but instead the CTG ECI Java methods were executed directly inside the WebSphere servlet engine. The CTG then utilized the CICS Universal Client ECI protocol to send the requests via LU6.2 into the target CICS region. The flow of control of in our Trader servlet application is illustrated in Figure 141.

![Figure 141. Windows NT CTG servlet application](image)

Our servlet was developed in Visual Age for Java and was called traderb2. It used the servlet API to respond to HTTP requests and returned HTML pages directly back to the requester. It called the CICS business logic program TRADERBL using the ECI methods supplied by the CICS Transaction Gateway. For further details on the Trader servlet, or to obtain the servlet code, refer to Appendix A, “The Trader application” on page 205.
Configuring WebSphere

We configured WebSphere on our two Web server machines, loire and malawi. The steps shown here are for the machine malawi but were identical on each machine.

First, we installed WebSphere AS Standard Edition V3.02 on the machine, followed by fixpack 1. We then re-booted the machine and started the Windows NT service IBM WS AdminServer. Following this, we started the WebSphere AS Administrator’s Console and we were presented with the window shown in Figure 142.

![WebSphere Standard Administrative Console](image)

Figure 142. WebSphere — task wizard 1

Our first task in deploying our servlet was to create a new Web application. This was achieved using the Tasks tab menu item, where we selected the Configure a Web application option under the Configuration menu.
We clicked the **green button** to and were presented with the screen illustrated in Figure 143, asking us to set the application name.

![Figure 143. WebSphere — task wizard 2](image)

We set the Web Application Name field to trader and also checked the box **Serve Servlets By Classname**. This option allowed us to use the servlet **autoinvoker** which automatically invokes a servlet, without needing to individually define each servlet. We then clicked **Next** to continue the dialog.
Next we selected **servlet engine**, by expanding the **Node** in the task wizard instructions (Figure 144).

![WebSphere task wizard](image)

**Figure 144. WebSphere — task wizard 3**

We clicked **Next** to continue the task wizard, and this took us to the next screen where we set the properties of our Web Application, as shown in Figure 145.
We named our Web application trader. We left the virtual host as the default of `default_host`, since we did not plan to create any other virtual hosts. We entered `/trader` as our URL for the Web Application Web Path. Since we had selected the WebSphere autoinvoker to dynamically serve our trader servlet, this meant that the URL which would be required to run our servlet `traderb2`, would be as follows:


We then clicked on the **Finished** button, which terminated the task configuration wizard, and updated WebSphere with our configuration.
Now we needed to deploy our servlet traderb2. We selected the Trader Web application under the Topology tab to show us the classpath in use. This is shown in Figure 146, and you can see that the classpath is set to `C:/WebSphere/AppServer/hosts/default_host/trader/servlets`.

![Image of WebSphere Standard Administrative Console](image)

Figure 146. WebSphere — task wizard 5

Since WebSphere does not create this directory that the classpath indicates, we had to do this manually. So we created a directory which matched the classpath and then we copied our servlet class file (`traderb2.class`) into this directory, so it could be served by WebSphere.
Finally, we had to add the CTG Java classes (ctgclient.jar and ctgserver.jar) to the Web Application Server classpath. Since the CTG Java classes utilize the JNI (to access the CICS Universal Client libraries), we had to configure the classpath in a different way than is usual. We updated the ‘Command line arguments’ of the Application Server as shown in Figure 147, with the additional argument:

```
-classpath D:\ctg\classes\ctgclient.jar;D:\ctg\classes\ctgserver.jar
```

Note that we also found that WebSphere would not allow us to use a path for these jar files that included the space character. Therefore we had to install the CTG in the directory `D:\CTG`, rather than in `C:\Program Files\IBM\CICS Transaction Gateway`, which is the default location.

![Figure 147. WebSphere — setting the classpath](image-url)
Finally, we had to update the path of the Web Application server, to enable the CTG native library (CTGJNI.DLL) to be located. To do this we added the environment variable PATH=D:\CTG\BIN to our Web application server; this is shown in Figure 148.

Note that when adding this environment variable, we found that you have to perform three steps that were not necessarily obvious. These were as follows:

1. Click on the **Environment** field.
2. Add the statement **PATH** as a Variable Name.
3. Add the statement **d:ctg\bin** as the Value.
4. Click the **Add** button.
5. Click the **OK** button.
6. Click the **Apply** button.
7. Restart the Default Server.
You might think that at this stage that we had performed all the necessary configuration steps of the Web Application server. However, we found out the hard way that we had to perform one last configuration step to enable WebSphere to work with our Network Dispatcher configuration. This was to add an alias, to enable WebSphere to respond to HTTP request destined for a different host than the WebSphere host. We selected the `default_host`, and under the Advanced tab, we added Aliases for the cluster address of Network Dispatcher (nassau). We added the hostname, the fully qualified hostname and the dotted decimal IP address (just to make sure we had all the bases covered). This is illustrated in Figure 149.

Without performing this step, we found that our Web servers continually gave HTTP error 404, since WebSphere was rejecting the requests and the Web server was trying to serve the servlet requests as if they were a standard request for a file owned by the Web server.

![Figure 149. WebSphere — adding aliases](image)

Now, at last, the Web application was ready to run.
The CICS Universal Client
In this section we give details about configuring the CICS Universal Client, which is used by the CTG to communicate with the CICS regions. A variety of network protocols are supported by the CICS Universal Client, but if you wish to communicate with a CICS/ESA system, you can only use SNA (LU6.2) or TCP62 (LU6.2 over IP). We chose SNA for our scenario.

Configuring SNA
We utilized the SNA support supplied with IBM Personal Communications V4.3 for Windows NT. Unfortunately, it is not possible to explain the full depths of SNA configuration in just a few paragraphs. However, below we have given a brief outline of our configuration of SNA on the machine loire.

Node
We configured our *Fully qualified CP name* as USIBMSC.SC02010.

Device
We configured the LAN device LANX_04 to use our token ring card.

Connection
We configured a Link station called VTAMLNK and entered the *Destination address* as the Token Ring card address of our VTAM Communications Controller. We set the *Block ID* and *Physical Unit ID* to the XID defined for our node in VTAM, and we set the *Adjacent Node CP name* to USIBMSC.SCG20 (the VTAM SSCP).

Mode
We configured a mode APPCMODE, with a *PLU mode session limit* and a *Maximum negotiable session limit* of 100, and with 99 *Minimum content winner sessions*.

Partner LU6.2
We configured partner LUs for all the CICS listener regions SCSCPWA1, SCSCPWA2, SCSCPWA3, SCSCPWA4, using USIBMSC.SCG20 as the *Fully qualified CP name*.

Local LU 6.2
We configured an independent Local LU named SC02010I.

The configuration was identical on the machine malawi, apart from the fact that we used the Local LU6.2 SC02009I, and the *Fully qualified CP name* of USIBMSC.SC02009 for the Node. For full details on configuring SNA for the CICS Universal Client, refer to the manual *CICS Universal Client for Windows Administration*, SC34-5549.
The CTG configuration tool
We used the new CTG configuration tool to configure the CICS Universal Client. This is started using the Configuration Tool in the IBM CICS Transaction Gateway folder. The first screen asks the names of the CICS server we wish to connect to (Figure 150). We entered the name of our first CICS listener region (SCSCPW1).

![Figure 150. CTG configuration tool — server definition](image)

We clicked the Next button, and then selected the SNA protocol (Figure 151).

![Figure 151. CTG configuration tool — protocol](image)
We then specified the Partner LU name (*USIBMSC.SCSCPWA1*), our Local LU name (*SC02010I*), and the Mode name (*APPCMODE*) — see Figure 152.

![Figure 152. CTG configuration tool — SNA definition](image)

We then repeated this process for our other three router CICS regions SCSCPWA2, SCSCPWA3 and SCSCPWA4. Having done this our basic server configuration was complete (Figure 153).

![Figure 153. CTG configuration tool — Server connections](image)
Configuring the CICS Universal Client Workload Manager

The next stage in our configuration was to configure the CICS Universal Client Workload Manager to distribute our ECI requests from the CICS Transaction Gateway across all four of our CICS listener regions. This enabled us to remove any dependency on an individual CICS listener region or OS/390 LPAR.

We started the CTG Configuration Tool and then selected the Workload Manager tab in the left-hand panel. We selected the Enable the Workload Manager checkbox, and the Round Robin algorithm checkbox, as shown in Figure 154. Round Robin is the more appropriate workload management technique, since the CICS Universal client Workload Manager does not have the ability to distribute work across CICS regions based on feedback from the regions (although it does have the ability to detect a failed CICS region).

![CTG configuration tool — Workload Manager settings](image)

The next stage was to add a program definition for the CICS program (TRADERB2) that we wished to workload manage (Figure 155). This enables the CICS Universal Client to modify the destination server for an ECI request to this program.
We selected the **Programs** tab in the left-hand panel and then clicked the **New Program Definition** button. We were then prompted to enter the program name. We entered TRADERB2 as shown in Figure 155, and then clicked **OK**.

![Figure 155. CTG configuration tool — adding a program](image)

For each of the CICS regions that we wished to include in this program’s workload definition, we selected the region in the right-hand panel, and then clicked the **<<** button to move each region to the left-hand panel (Figure 156).

![Figure 156. CTG configuration tool — program definitions](image)

With the program configured, we saved the configuration file, and restarted the CICS Universal Client using the command `cicscli /j` to activate our changes.

---

**194 Workload Management for Web Access to CICS**
10.1.3 CICS configuration

The configuration of our CICS regions for our Windows NT Trader servlet scenario was essentially the same as our OS/390 Trader servlet scenario described in 9.1.3, “Configuring CICS” on page 150. The only additional step was the LU6.2 autoinstall connection configuration. To configure LU6.2 connection autoinstall, we created a copy of the CBPS connection and sessions, and the DFHZATDY program from the CICS supplied group DFHAI62. We then modified the CBPS sessions definition to match our APPCMODE defined on Windows NT (see “Configuring SNA” on page 190) and installed this group. This is illustrated in Figure 157.

![Figure 157. CBPS sessions definition](image)

We also had to install the DFHISC and DFHCLNT CICS supplied groups, and we added the SIT parameter AIEXIT=DFHZATDY to enable LU6.2 connection autoinstall. We also added support for our new LU6.2 modegroup, by modifying adding the PARSESS and MODETAB statements to our VTAM APPL statements for all our CICS listener region, as shown in Figure 158.

![Figure 158. VTAM APPL definition](image)

**CICSPlex SM**

We used CICSPlex SM in goal mode to control the dynamic routing of our TRADERBL program from the listener regions to the AORs. This configuration was unchanged from that described in 9.1.4, “Configuring CICSPlex SM” on page 154.
10.2 Running our workload

Once we had configured our scenario we developed a workload to test the smooth running of this solution. This allows a Web user to buy and sell shares with one of four companies, or to query the number of shares held with those companies. Using a workload driver tool, we simulated 60 users all buying one share, 1000 times. For each “buy share” activity the traderb2 servlet made two ECI calls into CICS, one request to buy the shares, and another request to get the value of shares now held. The think time between each request was set to zero. This is illustrated in Figure 159.

As a reminder, the scenario used in our WebSphere NT Trader workload was invoked by specifying the following URL:

http://nassau/trader/servlet/traderb2

So for our workload of 60 users, we expected to see 6,000 total shares bought, and 12,000 calls to the dynamically routed program TRADERB2 in the listener regions, and 12,000 calls to TRADERBL in the AORs.
Network Dispatcher and WebSphere results

Figure 160 shows the weights assigned to loire (9.1.150.48) and malawi (9.1.150.43) over a 15 second period. These weights are gently oscillating, indicating slight fluctuations in HTTP ping response time. The smoothing index of 10 used in the Network Dispatcher configuration, ensured that the fluctuation was not too great. We found that for some workloads, if the smoothing index was left to default to 1, then the number of active connections varied considerably as Network Dispatcher switched between assigning all loads to loire and then to malawi.

Figure 161 shows the active connections for loire (9.1.150.43) and malawi (9.1.150.48) The number of connections rises from zero to approximately 30 per machine, corresponding to an even distribution of simulated users across the two machines. There is a slight oscillation before the balance of load begins to steady.
Figure 161. Network Dispatcher — active connections

Figure 162 shows the throughput for requests to the traderb2 servlet on the machine loire. These results were obtained using the WebSphere performance analyzer. The fluctuation in load tallies with the fluctuations reported by the Network Dispatcher utilities in Figure 160 and Figure 161.

Figure 162. WebSphere — performance analyzer
CTG/CICS Universal Client results

During the workload we used the Personal Communication SNA Node Operations tools to view the status of the SNA sessions bound from Windows NT to our CICS regions. Before we started the workload we restarted the SNA connection, and then the CICS client. This ensured that only the minimum number of sessions were bound before the workload started. Since each ECI request requires exclusive use of a SNA session for the duration of the ECI call, the number of Active Sessions for our mode (APPCMODE), tells us the maximum number of simultaneous requests to any one CICS region. In Figure 163, you can see the output from the machine malawi, this shows that the maximum number of active sessions varied from 9 to 11 for this run.

Note that this information could also have been derived from CICSPlex SM or from the CICS statistics, in a manner similar to how we extracted maximum socket usage for the CICS TCP/IP listener (see 8.2, “Running our workload” on page 131).

![Personal Communications SNA Node Operations](image)

**Figure 163. Personal Communications — active sessions**

CICS results

To check program usage on our CICS regions, we used the CICSPlex SM Web User Interface (WUI) to investigate how many times the program TRADERBL had been executed in the AORs, and how many times the DFHCNV code page conversion templates had been referenced in the listener regions. These results are illustrated in Figure 164 and Figure 166.

Since the DFHCNV code page conversion template is referenced once for every ECI request, the usage count is a direct indication of how many ECI requests were made to any given listener region. Note that we discarded the TRADERBL program definition before we ran the workload, so the count starts from zero. It is no possible, however, to discard the definition for DFHCNV, since this is a CICS supplied definition.
### Figure 164. CICSPlex SM WUI — DFHCNV calls before workload

![Image of CICSPlex SM WUI showing DFHCNV calls before workload](image1.png)

### Figure 165. CICSPlex SM WUI — DFHCNV calls after workload

![Image of CICSPlex SM WUI showing DFHCNV calls after workload](image2.png)

---

200 Workload Management for Web Access to CICS
These results are summarized in Table 18.

Table 18. WebSphere NT scenario — CICS program usage

<table>
<thead>
<tr>
<th>Region</th>
<th>DFHCNV</th>
<th>TRADERBL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSCPW1</td>
<td>2,988</td>
<td></td>
</tr>
<tr>
<td>SCSCPW1</td>
<td>2,986</td>
<td></td>
</tr>
<tr>
<td>SCSCPW3</td>
<td>3,022</td>
<td></td>
</tr>
<tr>
<td>SCSCPW4</td>
<td>2,994</td>
<td></td>
</tr>
<tr>
<td>SCSPAA1</td>
<td></td>
<td>6,205</td>
</tr>
<tr>
<td>SCSPAA4</td>
<td></td>
<td>5,795</td>
</tr>
</tbody>
</table>

These results clearly show us that 12,000 calls were made to the TRADERB2 program in the listener regions (SCSCPW1, SCSCPW2, SCSCPW3, SCSPWA4), and also to the TRADERBL program in the AORs (SCSPAA1, SCSPAA4).

Trader results

Finally, we checked the number of shares purchased for our given test user, by using a Web browser and the traderb2 servlet. This confirmed that 6,000 shares had been purchased.
10.2.1 Failover scenarios

In order to really test the usefulness of our workload management configuration, we ran the following series of failover tests. In each test we caused the outage of one particular software component, during the running of a workload consisting of 60 simulated users each invoking 1,000 buy share requests (a total of 6,000 requests). Following completion of the workload we analyzed the results for each component, to see where failures occurred.

10.2.1.1 Stopping a Web server

In this scenario we stopped the IBM HTTP server Windows NT service on the machine loire, during the running of the workload, as illustrated in Figure 167.

![Diagram](image)

Figure 167. WebSphere NT — stopping a Web server

The results are shown in Table 19.

Table 19. WebSphere NT results — stopping a Web server

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>5,954 pages complete, 396 failed connections 46 early server closes,</td>
<td>5,954 (6,000)</td>
<td>46</td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 2,956  SCSCPWA2 - 2,983  SCSCPWA3 - 3,023  SCSCPWA4 - 2,984</td>
<td>11,946 (12,000)</td>
<td>54</td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - 5,746  SCSCPAA4 - 6,210</td>
<td>11,956 (12,000)</td>
<td>44</td>
</tr>
<tr>
<td>Trader</td>
<td>5,978 shares traded</td>
<td>5,978 (6,000)</td>
<td>22</td>
</tr>
</tbody>
</table>

These results indicate the ability of the Network Dispatcher to detect a failed Web server and to remove the traffic away from that machine. The number of failures could have been reduced by decreasing the update interval for the Network Dispatcher HTTP advisor from the default of 7 seconds.
10.2.1.2 Crash of a Web server machine
In this scenario we terminated the WebSphere machine loire, by switching off
the PC during the running of the workload. This is illustrated in Figure 168.

Figure 168. WebSphere NT — cancel of a Web server

The results are shown in Table 20.

Table 20. WebSphere NT results — cancel of a Web server

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>5,987 pages complete 13 early server closes</td>
<td>5,987 (6,000)</td>
<td>13</td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 2,998 SCSCPWA2 - 2,990 SCSCPWA3 - 2,998 SCSCPWA4 - 2,990</td>
<td>11,976 (12,000)</td>
<td>24</td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - 5,106 SCSCPAA4 - 6,870</td>
<td>11,976 (12,000)</td>
<td>24</td>
</tr>
<tr>
<td>Trader</td>
<td>5,992 shares traded</td>
<td>5,992 (6,000)</td>
<td>8</td>
</tr>
</tbody>
</table>

These results indicate the ability of Network Dispatcher to very quickly detect
a machine failure and to route work away from the failed node. In this
situation only 13 failures were noted by the HTTP driver, and only 8 shares
failed to be traded in the CICS AORs, even though one of the Web server
machines crashed.
10.2.1.3 Restart of a listener region
In this scenario we started the workload with only 3 of the 4 CICS listener regions active. We then started the fourth listener region (SCSPWA4) during the running of the workload and analyzed the results to see if it was brought on-line once it became active. This is illustrated in Figure 169.

![Figure 169. webSphere NT — restart of failed listener region](image)

The results are shown in Table 21.

**Table 21. WebSphere NT results — restart of failed listener region**

<table>
<thead>
<tr>
<th>Component</th>
<th>Results</th>
<th>Successes (expected)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP driver</td>
<td>6,000 pages complete</td>
<td>6,000 (6,000)</td>
<td>none</td>
</tr>
<tr>
<td>CICS listener regions (DFHCNV calls)</td>
<td>SCSCPWA1 - 3,766</td>
<td>12,003 (12,000)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>SCSCPWA2 - 3,790</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA3 - 3,774</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCSCPWA4 - 6,73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS AORs (TRADERBL calls)</td>
<td>SCSCPAA1 - 5,955</td>
<td>12,000 (12,000)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>SCSCPAA4 - 6,045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trader</td>
<td>6,000 shares</td>
<td>6,000 (6,000)</td>
<td>none</td>
</tr>
</tbody>
</table>

These results indicate the ability of the CICS Universal Client Workload Manager to bring an inactive or failed region back on-line without affecting the success of any of the running transactions. The 3 extra calls to DFHCNV in the listener regions (12,003 as opposed to 12,000) are due to 3 calls to DFHCNV made by a CICS region on startup.
Appendix A. The Trader application

Our sample green-screen application is called Trader. Trader allows authenticated users to trade shares, that is, to buy and sell shares in a given group of companies, as well as obtaining real-time quotes on the value of their current holdings. Trader has been developed as a sample as part of an IBM CICS Web-enablement service offering.

Sample code and templates required to Web-enable the Trader using all of the technologies documented in this redbook are available as additional materials. For download instructions, refer to Appendix B, “Using the additional material” on page 221.

We have been using Trader as our sample application throughout this IBM Redbook for our Workload management study. It provides a sample of a traditional COBOL/VSAM application, with Web-enabled version that can run via CICS Web Support, the 3270 Web bridge, or as a servlet using the CICS Transaction Gateway on any supported platform.

A.1 3270 Trader

Trader, written in COBOL, uses the VSAM access method for file access and the CICS 3270 BMS programming interface. It is a pseudo-conversational application, meaning that a chain of related non-conversational CICS transactions is used to convey the impression of a "conversation" to the users as they go through a sequence of screens that constitute a business transaction. A non-conversational CICS transaction has one input and one output, so no task waits for user input as the user examines a screen and enters responses into it. CICS provides several facilities for passing information about the current state of the business transaction forward from one task to another. The most commonly used is the COMMAREA data structure which can be associated with the terminal.

At each step, the application presents a set of options. The user makes a choice, then presses the required key in order to send their selections back to the application. The application performs the necessary actions based on the user’s choice and presents the results together with any possible new options. The application has a strict hierarchical menu structure which allows the user to return to the previous step by using the PF3 key. The application consists of two modules: TRADERPL, which contains the 3270 presentation logic; and TRADERBL, which contains the business logic.
3270 application flow

In this section we describe the 3270 Trader application in more detail:

1. The program TRADERPL is invoked on a 3270 capable terminal by entering the initial CICS transaction identifier (TRAD). TRADERPL calls TRADERBL, passing an inter-program COMMAREA of 400 bytes. TRADERBL expects the COMMAREA to contain a request type and associated data. There are 3 request types: Get_Company to return a company list, Share_Value to return a list of share values, or Buy_Sell to buy or sell shares. In this step the request type is Get_Company.

When TRADERBL receives a Get_Company request, it browses the company file and returns the first four entries to TRADERPL. At this point the user has not entered any request, but the application assumes that a Get_Company request will be following. TRADERPL then sends the signon display (T001 shown in Figure 170), which prompts for a userid and password. The list of companies is stored in the COMMAREA associated with the terminal when the TRAD transaction ends, so that it will be available at the next task in the pseudo-conversational sequence.

![Figure 170. Trader signon display](image)

2. The next transaction invokes TRADERPL, which receives the signon display (T001) and the saved COMMAREA from step 1. Using the company data acquired in step 1, TRADERPL sends the company selection display (T002), the format of which is shown in Figure 171. TRADERPL then returns, specifying the next transaction to run and the associated COMMAREA.
3. The user selects the company to trade from the *Company Selection* display, and presses Enter. The program TRADERPL is invoked and sends the *Options display* (T003, shown in Figure 172) to the terminal. The user can now decide whether to buy, sell, or get a new real-time quote. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA.

![Figure 171. Company selection display](image1)

![Figure 172. Options menu display](image2)
4. The user then selects option 1 and presses the Enter key. TRADERPL is invoked and determines that the user’s request is a Share_Value request type. TRADERPL calls TRADERBL, passing the request type and the company selected earlier. TRADERBL reads the customer file to determine how many shares are held, then reads the company file to determine the price history, and returns the information to TRADERPL. TRADERPL uses this data to build a Real-Time Quote display (T004) as illustrated in Figure 173. This display shows the recent history of share values for the company chosen, the number of shares held with this company, and the total value of these shares. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA data.

<table>
<thead>
<tr>
<th>Share Trading Demonstration</th>
<th>Share Trading Manager: Real-Time Quote</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Name: TRADER</td>
<td>Company Name: IBM</td>
</tr>
<tr>
<td>Share Values:</td>
<td>Commission Cost:</td>
</tr>
<tr>
<td>NOW: 00163.00</td>
<td>for Selling: 015</td>
</tr>
<tr>
<td>1 week ago: 00157.00</td>
<td>for Buying: 010</td>
</tr>
<tr>
<td>6 days ago: 00156.00</td>
<td></td>
</tr>
<tr>
<td>5 days ago: 00159.00</td>
<td></td>
</tr>
<tr>
<td>4 days ago: 00161.00</td>
<td></td>
</tr>
<tr>
<td>3 days ago: 00160.00</td>
<td></td>
</tr>
<tr>
<td>2 days ago: 00162.00</td>
<td></td>
</tr>
<tr>
<td>1 day ago: 00163.00</td>
<td></td>
</tr>
<tr>
<td>Number of Shares Held: 0100</td>
<td>Value of Shares Held: 000000000.00</td>
</tr>
</tbody>
</table>

Figure 173. Real-time quote display

5. The user now presses PF3 to go back to the options menu. TRADERPL is invoked and sends the Options display (T003) to the terminal (repeating the actions of step 3), and returns, specifying the next transaction to run and the associated COMMAREA data.

6. The user now requires to purchase shares, so selects option 2 and presses the Enter key. Program TRADERPL receives map T003 and determines that the user wants to buy shares, and sends the Shares-Buy display (T005) shown in Figure 174. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA.
7. Program TRADERPL receives the T005 screen and builds a *Buy_Sell* request COMMAREA which is passed to program TRADERBL. TRADERBL reads the company file and then performs a READ for UPDATE and REWRITE to the customer file to update the customers share holdings. The success of the request is returned to TRADERPL in the COMMAREA, and TRADERPL sends the *Options display* (T003) reporting the successful buy to the user. TRADERPL returns, specifying the next transaction to run and the associated COMMAREA.

8. Next the user checks his share holdings by repeating step 4.

9. The user returns to the options screen by repeating step 5.

10. The business transaction is completed by the user pressing PF12, which performs a *SEND TEXT* to write a message to the terminal reporting the session is complete. TRADERPL then executes the final *RETURN* command. No COMMAREA is specified because the pseudo-conversation is over, and there is no conversation state data to retain.
A.2 Trader via CWS and a Web-aware converter

In this section we describe how the Trader application functions when Web-enabled via the facilities of CWS. The new Web-aware presentation logic is implemented in a converter module called TRADERCV, and the HTTP data streams are manipulated using the CICS WEB and DOCUMENT API. This converter can be used using a CWS direct connection or the CICS WebServer Plugin.

The flow of CICS tasks is illustrated in Figure 175, and described in the following paragraphs.

![Figure 175. Trader application flow using CWS and Web-aware presentation logic](image-url)
1. The URL hostname:port/tradercv/cwba/traderbl is entered at the browser. CWS receives the initial HTTP GET for TRADERBL from the browser, with the converter program TRADERCV specified in the request. A CWXN Web attach transaction is started and handles all further HTTP requests for this business transaction, using a persistent HTTP connection. TRADERCV is invoked, which builds the signon page HTML using the CICS DOCUMENT and WEB API and sends it back to the browser; this is illustrated in Figure 176.

![Figure 176. CWS Trader — signon page](image)

The user enters a userid and password for verification, and clicks the Submit button.
2. The Web browser does an HTTP POST of the signon form. The request is passed to TRADERCV, which calls TRADERBL passing a COMMAREA as input. TRADERBL verifies the userid and password, reads the company and customer files and returns the result to TRADERCV via the COMMAREA. TRADERCV returns the *company selection* HTML page, as shown in Figure 177.

![Figure 177. CWS Trader — company selection page](image)

The user chooses a company from the drop down box, and clicks the **Submit** button.
3. The Web browser does a POST of the company selection form. TRADERCV receives this data, and calls TRADERBL via the COMMAREA.

4. TRADERBL browses the company and customer files, builds the buy-sell quote page, and returns the quote to the browser, as shown in Figure 178.

![Share Trading Demonstration](image)

Figure 178. CWS Trader — buy-sell quote page

The user enters the number of shares to buy, selects the Buy Shares radio button and clicks the Submit button.

5. The Web browser does an HTTP POST of the completed buy-sell quote page with the number of shares to buy. The request is passed to TRADERCV, which calls TRADERBL passing a COMMAREA. TRADERBL updates the share holdings in the customer file, and calculates the value of the updated holdings. TRADERCV sends the buy-sell quote page with the value of the new share holdings.
6. The Web browser does an HTTP POST when the user clicks the **End Trader** radio button. TRADERCV returns the *Trader Complete* page, as shown in Figure 179.

![Figure 179. CWS Trader — The Trader Complete page](image_url)
Appendix A. The Trader application

A.3 Trader via a servlet and the CTG

In this section we describe how the Trader application functions when Web-enabled via the facilities of a servlet and the CICS Transaction Gateway Java classes. The new presentation logic is implemented in the servlet, and ECI calls are made to the business logic in the program TRADERBL.

The flow of CICS tasks is illustrated in Figure 180, and described in the following paragraphs.

![Figure 180. Trader application flow using servlet and CTG](image-url)
1. The URL hostname:port//servlet/traderb2 is entered at the browser. The Web server receives the initial HTTP GET for the servlet and passes it to the configured WebSphere Application Server. The servlet notices that no session token is included, so returns the initial signon page, as illustrated in Figure 181.

![Servlet Trader — signon page](image)

Figure 181. Servlet Trader — signon page

The user enters a userid and password for verification, and clicks the Login button.
2. The Web browser does an HTTP POST of the signon form. The request is passed back to the servlet, which opens a connection to the Java Gateway and makes an ECI call to the CICS program TRADERBL, passing a COMMAREA as input. TRADERBL verifies the userid and password passed in the COMMAREA, reads the company and customer files and returns the result to the servlet via the COMMAREA. The servlet returns the company selection HTML page, as shown in Figure 182.

![Share Trading Demonstration](image)

Figure 182. Servlet Trader — company selection page

The user chooses a company from the drop-down box, and clicks the Select a Company button.
3. The Web browser does a POST of the company selection form. The servlet receives this data, and issues an ECI call to TRADERBL with the specified company in the COMMAREA. TRADERBL browses the company and customer files, and returns the information to the servlet. The servlet builds the *buy-sell quote page*, and returns the current quote to the browser, as shown in Figure 183.

![Figure 183. Servlet Trader — buy-sell quote page](image)

The user enters the number of shares to buy, selects the **Buy Shares** radio button, and clicks the **Trade** button.
4. The Web browser does an HTTP POST of the completed *buy-sell quote page* with the number of shares to buy. The request is passed to the servlet, which issues an ECI call to TRADERBL passing a COMMAREA containing the number of shares to buy. TRADERBL updates the share holdings in the customer file, and calculates the value of the updated holdings and returns this to the servlet. The servlet sends the *buy-sell quote page* with the value of the new share holdings, which is the same as the page shown in Figure 183, but with the updated values.

5. The user then clicks the **Exit** button. The Web browser does an HTTP POST of the form. The servlet receives the form, and returns the *Trader Complete page*, as shown in Figure 184.

![Share Trading Demonstration](image)

*Figure 184. Servlet Trader — The Trader complete page*
Appendix B. Using the additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

B.1 Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246118

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the redbook form number, SG24-6118.

B.2 Using the Web material

The additional Web material (SG246118.zip) that accompanies this redbook includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallCICS_Readme.txt</td>
<td>Readme for CallCICS servlet</td>
</tr>
<tr>
<td>CallCICS.java</td>
<td>Sample CTG ECI test servlet Java source</td>
</tr>
<tr>
<td>Base64.java</td>
<td>Base64 decode/encoder Java source</td>
</tr>
<tr>
<td>CallCICS.class</td>
<td>Sample CTG ECI test servlet Java class</td>
</tr>
<tr>
<td>Base64.class</td>
<td>Base64 decode/encoder Java class</td>
</tr>
<tr>
<td>ECIPROG.TXT</td>
<td>Sample CICS COBOL application</td>
</tr>
<tr>
<td>DFHXCURM_readme.txt</td>
<td>Readme for DFHXCURM sample</td>
</tr>
<tr>
<td>ctgxcurm.txt</td>
<td>Sample DFHXCURM</td>
</tr>
<tr>
<td>cpsmget.txt</td>
<td>CICSPlex SM module</td>
</tr>
<tr>
<td>xcurmjcl.txt</td>
<td>JCL to link above modules</td>
</tr>
<tr>
<td>Trader_ReadMe.txt</td>
<td>Readme for Trader samples</td>
</tr>
<tr>
<td>deltsqs.txt</td>
<td>TS deletion program (COBOL)</td>
</tr>
<tr>
<td>traderbl.txt</td>
<td>TRADERBL application (COBOL)</td>
</tr>
<tr>
<td>tradercv.txt</td>
<td>Trader CWS converter (COBOL)</td>
</tr>
<tr>
<td>traderpl.txt</td>
<td>TRADERPL application (COBOL)</td>
</tr>
<tr>
<td>tradwbsr.txt</td>
<td>Trader TS state management program (COBOL)</td>
</tr>
</tbody>
</table>
newtradb.txt Mapset for trader 3270 application
tradercodata.txt Trader company file data
tradercojcl.txt JCL for creating company file
tradercujcl.txt JCL for creating customer file
traderrdo.txt RDO definitions for Trader

tradbann.html Trader CWS HTML template
tradcomp.html Trader CWS HTML template
tradend.html Trader CWS HTML template
tradfoot.html Trader CWS HTML template
tradhead.html Trader CWS HTML template
tradqbs.html Trader CWS HTML template
tradsign.html Trader CWS HTML template

StateData.java Trader servlet Java source
trader.java Trader servlet Java source
traderb2.java Trader servlet Java source
TraderBase.java Trader servlet Java source
TraderCommarea.java Trader servlet Java source
StateData.class Trader servlet Java class
trader.class Trader servlet Java class
traderb2.class Trader servlet Java class
TraderBase.class Trader servlet Java class
TraderCommarea.class Trader servlet Java class

B.2.1 System requirements for downloading the Web material

The following system configuration is recommended:

**Hard disk space:** 1 MB minimum
**Operating System:** Windows NT or 95
**Processor:** Intel 386 or higher
**Memory:** 16 MB

B.2.2 How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. This will create three directories, CallCICS, Trader, and DFHXCURM, each containing a readme file with further instructions.
Appendix C. Special notices

This publication is intended to help I/T architects and CICS system programmers to understand, plan for, and implement workload management solutions when Web-enabling legacy CICS applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by CICS Transaction Server, or OS/390 WebSphere Application Server, CICS Transaction Gateway, or Host On-Demand. See the PUBLICATIONS section of the IBM Programming Announcement for these products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

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 Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been
reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

- AIX
- CICS
- CICS/ESA
- CICS TS
- CICS Transaction Server
- CICSPlex
- DB2
- Domino
- Language Environment
- Lotus
- MVS/ESA
- MQSeries Open Edition
- OS/2
- Redbooks
- Redbooks Logo
- OS/390
- Parallel Sysplex
- RACF
- SAA
- S/390
- SecureWay
- SP
- System/390
- TXSeries
- VisualAge
- VTAM
- WebSphere

The following terms are trademarks of other companies:

- Tivoli
- Manage.
- Anything.
- Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both.
- In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

- C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

- CORBA is a trademark of the Object Management Group, Incorporated.

- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.
Appendix D. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 231.

- A Performance Study of Web Access to CICS, SG24-5748
- CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge, SG24-5480
- IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide, Volume 1: Configuration and Routing, SG24-5227
- IBM Web-to-Host Integration Update, SG24-5237
- Java Application Development for CICS, SG24-5275
- IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing, SG24-2149
- OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390, SG24-5603
- OS/390 e-business Infrastructure: IBM WebSphere Application Server 1.2 Customization and Usage, SG24-5604
- OS/390 Work Load Manager Implementation and Exploitation, SG24-5326
- Revealed! Architecting Web Access to CICS, SG24-5466
- Revealed! CICS Transaction Gateway with More CICS Clients Unmasked, SG24-5277
- TCP/IP in a Sysplex, SG24-5235
- WebSphere Scalability: WLM and Clustering Using WebSphere Application Server Advanced Edition, SG24-6153
D.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Collection Kit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System/390 Redbooks Collection</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>IBM Networking Redbooks Collection</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>IBM Transaction Processing and Data Management Redbooks Collection</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>IBM Lotus Redbooks Collection</td>
<td>SK2T-8039</td>
</tr>
<tr>
<td>Tivoli Redbooks Collection</td>
<td>SK2T-8044</td>
</tr>
<tr>
<td>IBM AS/400 Redbooks Collection</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>IBM Netfinity Hardware and Software Redbooks Collection</td>
<td>SK2T-8046</td>
</tr>
<tr>
<td>IBM RS/6000 Redbooks Collection</td>
<td>SK2T-8043</td>
</tr>
<tr>
<td>IBM Application Development Redbooks Collection</td>
<td>SK2T-8037</td>
</tr>
<tr>
<td>IBM Enterprise Storage and Systems Management Solutions</td>
<td>SK3T-3694</td>
</tr>
</tbody>
</table>

D.3 Other resources

These publications are also relevant as further information sources:

- *CICS External Interfaces Guide*, SC33-1944
- *CICS Internet Guide*, SC34-5445
- *CICS Performance Guide*, SC33-1699
- *CICS System Definition Guide*, SC33-1682
- *CICS Transaction Affinities Utility Guide*, SC33-1777
- *CICS Universal Client for Windows Administration*, SC34-5449
- *CICSplex SM Web User Interface Guide*, SC34-5403
- *Communicating from CICS on System/390*, SC33-1697
- *HTTP Server Planning, Installing, and Using*, SC31-8690
- *SNA Network Implementation Guide*, SC31-8563
- *IP Configuration Reference*, SC31-8726
- *IP Planning and Migration Guide*, SC31-8512
D.4 Referenced Web sites

These Web sites are also relevant as further information sources:

  Network Dispatcher whitepaper

- http://www.cookiecentral.com
  Information on HTTP cookies

  IBM glossary of computing

  CICS Web/selection guide whitepaper

  Internet Engineering Task Force, request for comments repository

  S/390 e-business performance page

- http://www.w3.org/Security/Faq/wwwsf7.html#Q66
  The World Wide Web Security FAQ repository

- http://www.whatis.com
  IT-specific on-line encyclopedia
How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** [ibm.com/redbooks](http://ibm.com/redbooks)
  
  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**
  
  Send orders by e-mail including information from the IBM Redbooks fax order form to:

  **e-mail address**

  In United States or Canada: pubscan@us.ibm.com
  Outside North America: pubscan@ibmlink.ibm.com

  Contact information is in the “How to Order” section at this site:


- **Telephone Orders**

  United States (toll free): 1-800-879-2755
  Canada (toll free): 1-800-IBM-4YOU
  Outside North America: Contact information is in the “How to Order” section at this site:


- **Fax Orders**

  United States (toll free): 1-800-445-9269
  Canada: 1-403-267-4455
  Outside North America: Fax phone number is in the “How to Order” section at this site:


This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

---

**IBM Intranet for Employees**

IBM Redbooks fax order form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Company</th>
<th>Address</th>
<th>City</th>
<th>Postal code</th>
<th>Country</th>
<th>Telephone number</th>
<th>Telefax number</th>
<th>VAT number</th>
</tr>
</thead>
</table>
| Invoice to customer number
| Credit card number |

<table>
<thead>
<tr>
<th>Credit card expiration date</th>
<th>Card issued to</th>
<th>Signature</th>
</tr>
</thead>
</table>

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.
Glossary


An excellent glossary of Internet and other computing terms is available at:
http://www.whatis.com

abend. Abnormal end of task, usually referring to a CICS task.

applet. A small Java program, downloaded from a Web server to a Web browser. Applets differ from fully fledged Java applications in that they can be restricted from performing certain operations on the local computer.

application-owning region (AOR). A CICS region in a MRO environment that “owns” the CICS applications, and invokes them on behalf of remotely attached terminal (or Web) users. See also TOR and listener region.

Application Programming Interface (API). Application programming interface. A set of calling conventions defining how a service is invoked through a software package.

Advanced Program-to-Program communication (APPC). An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

business logic interface (BLI). An externally callable interface provided by CICS Web support. It allows a client to invoke the business logic in a CICS application. It is implemented by the module DFHWBBLI. It provides a mechanism for implementing Web-aware presentation logic in the “converter”. The converter provides Decode and Encode routines to receive and send the HTTP presentation logic.

browser. An application that displays World Wide Web documents, usually referred to as a Web browser.

Central Electronic Complex (CEC). This is the physical machine that contains main storage (memory), central processing units and connections to devices. Used primarily in the S/390 environment, and also known as Central Processing Complex.

Central Processing Unit (CPU). Known also as an engine or processor, this is the part of the computer that executes the program instructions. There may be one or many CPUs in an S/390 CEC. Each CPU in the CEC may access the main storage (memory) in that CEC. If there are multiple CPUs in a CEC, then multiprocessing (or simultaneous execution of two threads of control) is possible.

Customer Information Control System (CICS). A distributed on-line transaction processing system designed to support a network of many terminals. The CICS family of products is available for a variety of platforms ranging from a single workstation to the largest mainframe.

CICS Transaction Gateway (CTG). A set of software components that provide the ability for CICS transaction and programs to be invoked from Java programs.

CICS Web support (CWS). A set of resources supplied with CICS TS V1.3 that provide CICS with functionality similar to a real Web server, and allows both CICS programs and 3270 transactions to be invoked from the Internet. See also CICS Web Interface.

CICS Web Interface (CWI). A set of resources supplied with CICS/ESA V4.1 and CICS TS V1.2 that allows CICS programs to be invoked from the Internet. Superseded by CICS Web support.

client. As in client/server computing, the application that makes requests to the server and, often, handles the interaction necessary with the user.

client-server computing. A form of distributed processing, in which the task required to be processed is accomplished by a client portion that requests services and a server portion that fulfills those requests. The client and server
remain transparent to each other in terms of location and platform. See client and server.

**COMMAREA.** A buffer or data area used by a CICS program to hold input and output data when LINKed (called) as a subroutine from another CICS program.

**commit.** An action that an transaction processing monitor takes, to make permanent the changes it has made to recoverable resources during a logical unit of work.

**Common Connector Framework (CCF).** IBM’s common client programming model for connectors. These interfaces allow VisualAge for Java’s Enterprise Access Builder for transactions to easily build Java applets or servlets to access programs or transactions in a CICS region. The CCF also provides a common infrastructure programming model for connectors, which gives a component environment such as WebSphere a standard view of a connector.

**Common Gateway Interface (CGI).** A defined standard that describes how a Web Server can communicate with another application (the CGI program) on the same machine. Usually a CGI program is a small program that takes data from a Web server and does something with it, like execute a database query.

**Common Object Request Broker Architecture (CORBA).** The Object Management Group’s (OMG) standard for communication between client and server ORBs.

**conversational.** A communication model where two distributed applications exchange information by way of a conversation; typically one application starts (or allocates) the conversation, sends some data, and allows the other application to send some data. Both applications continue in turn until one decides to finish (or deallocate). The conversational model is a synchronous form of communication. See also pseudo-conversational.

**cookie.** A piece of information sent by a Web Server to a Web browser that the browser software is expected to save and to send back to the Server whenever the browser makes additional requests from the Server. Depending on the configuration, the browser may or may not accept the cookie. Cookies can be stored on the hard disk of the Web browser and can be reused until they expire. Cookies often contain identification information such as registration information, or user preferences.

**Coupling Facility (CF).** Is a special OS/390 logical partition that provides high-speed caching, list processing, and locking functions between systems in a S/390 Parallel Sysplex. The coupling facility can run in an OS/390 LPAR or in a dedicated Coupling Facility processor.

**Cryptographic Coprocessor Feature.** An additional hardware unit for a S/390 system that contains dual cryptographic module chips protected by tamper-detection circuitry. It can be used to off-load cryptographic operations from the main processors.

**distributed program link (DPL).** This enables an application program executing in one CICS system to link to (or call) a program in a different CICS system. The linked-to program executes and returns a result to the linking program in a buffer known as a COMMAREA. This function is similar to a remote procedure call (RPCs) as provided by client-server architectures.

**Distributed Transaction Processing (DTP).** This enables a transaction running in one CICS system to communicate synchronously with transactions running in other system using the LU6.2 protocol. The transactions are designed and coded specifically to communicate with each other.

**Domain Name System (DNS).** System for resolving TCP/IP hostname to IP addresses, and vice versa.

**External Call Interface (ECI).** An application programming interface (API) provided by the CICS client that enables an application on a workstation to call a CICS program as a subroutine. The client application communicates with the server CICS program using a data area called a COMMAREA.

**External CICS Interface (EXCI).** An application programming interface (API) provided by CICS on OS/390 that enables a non-CICS application
running on OS/390 to call a CICS program as a subroutine. It is similar in function to the ECI but is available only on OS/390.

**External Presentation Interface (EPI).** An application programming interface (API) provided by the CICS Client that allows a 3270 based CICS transaction to be driven programatically.

**External Security Interface (ESI).** An application programming interface (API) provided by the CICS Universal Client that utilizes the APPC PEM support in a CICS region to verify and update userids and passwords.

**External Security Manager (ESM).** An external security product, such as RACF, used by CICS for authentication and authorization of users.

**extranet.** A combination of TCP/IP networks of different companies connected with a secure connection, perhaps using Virtual Private Network technology (VPN).

**form.** Part of a HTML document that allows users to enter data.

**function shipping.** A CICS intersystem communication protocol that enables an application program running in one CICS system to access resources owned by another CICS system. In the resource-owning system, a mirror transaction is initiated to perform the necessary operation; for example, to access CICS files or temporary storage, and to reply to the requester.

**gateway.** Software that transfers data between normally incompatible applications, or between networks.

**Graphical user interface (GUI).** A style of user interface that replaces the character-based screen with an all-points-addressable, high-resolution graphics screen. Windows display multiple applications at the same time and allow user input by means of a keyboard or a pointing device such as mouse, pen, or trackball.

**Go Webserver Application Programming Interface (GWAPI).** API provided by the Domino Go Webserver for OS/390.

**Hierarchical File System (HFS).** OS/390 UNIX System Services filing system.

**host.** (1) In a computer network, a computer providing services such as computation, database access, and terminal control functions. (2) In a multiple computer installation, the primary or controlling computer. Often used to refer to by an IBM S/390 processor.

**Host Access Class Library (HACL).** This is provided by IBM's Host On-Demand software as a set of Java classes and methods that allow the development of platform-independent applications that can access host information (such as CICS 3270 terminals) at the data stream level.

**Host On-Demand (HOD).** IBM's terminal emulation software, downloadable "on-demand" from a Web server.

**hypertext.** Text that activates connection to other documents when selected.

**Hypertext Markup Language (HTML).** Standard scripting language used to create hypertext documents. Widely used for documents on the World Wide Web.

**Hypertext Transmission Protocol (HTTP).** Standard Internet client/server communications protocol.

**Interface Definition Language (IDL).** A definition language used to define an interface between client and server objects, including the operations that can be performed by the server. Used by amongst others, the CORBA standard and the DCE protocol.

**intranet.** An internal Local Area Network, private to a corporation or other organization.

**internet.** A collection of interconnected networks, compare with the Internet.

**Internet.** The global TCP/IP network that forms the backbone of the World Wide Web.

**Internet Connection Application Programming Interface (ICAPI).** OS/390 Web server application programming interface, now replaced by the GWAPI.

**Internet Inter ORB Protocol (IIOP).** 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.

**intersystem communication (ISC).**
Intercommunication between separate CICS regions (systems) by means of SNA networking facilities or the application-to-application facilities of VTAM. ISC can also be used to connect other APPC based system such as the CICS Universal Client or native APPC applications. Compare with multi-region operation (MRO).

**Java Virtual Environment (JVM).** The virtual environment within which a Java application runs. The JVM provides for such features as platform neutrality and multi-threading.

**Java Native Interface (JNI).** Interface provided by the Java language to enable native (non-Java) code to be invoked.

**listener region.** A CICS region that listens for incoming requests and routes them onto to an attached application-owning regions. A CICS system can listen for request from HTTP or CORBA clients using the CICS TCP/IP listener, from LU6.2 clients using CICS ISC support, or from other OS/390 address spaces (such as the OS/390 Web server) using the EXCI protocol. See also application-owning region and terminal-owning region.

**Local Area Network (LAN).** A localized computer network usually based on ethernet or token-ring technology. See also intranet and WAN.

**Logical Partition (LPAR).** A LPAR is a logical subset of the S/390 CEC hardware. The CEC resources, CPUs and main memory, can be shared between LPARs. Each LPAR is capable of running an instance, or image, of the OS/390 operating system.

**LU type 6.2 (LU 6.2).** A type of SNA logical unit, that offers multiple (parallel), half duplex sessions between independent (peer to peer) devices. Used extensively by CICS intersystem communication. APPC is the application programming interface provided by an LU6.2 logical unit.

**Multi Region Operation (MRO).**
Intercommunication between CICS regions in the same processor (or sysplex) without the use of SNA network facilities. This allows several CICS regions to communicate with each other, and to share resources such as files, terminals, temporary storage, and so on. Contrast with intersystem communication.

**Network Address Translation (NAT).** A protocol used to provide a mapping between internal IP addresses and external or public (globally unique) IP addresses. Often used when connecting an intranet to the Internet. Since the internal addresses are not advertised outside the intranet, NAT can be used when they are private (globally ambiguous) addresses, or when they are public (globally unique) addresses that a company wishes to keep secret.

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

**Object Request Broker (ORB).** 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 interfaces.

**On-line Transaction Processing (OLTP).**
A style of computing that supports interactive applications in which requests submitted by terminal users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. An on-line transaction processing system supervises the sharing of resources to allow efficient processing of multiple transactions at the same time.

**Operating System 390 (OS/390).** IBM’s operating system for the S/390 platform. Previously known as MVS.

**Parallel Sysplex.** This is a sysplex that uses one or more coupling facilities.

**pipe.** An EXCI pipe is a one-way communication path between a sending client process and a receiving CICS region. Each pipe maps onto one CICS MRO session. See also EXCI and MRO.
proxy. A software gateway between connecting networks that allows communication between the two networks, by acting as both a client and a server. A popular usage of a proxy is a HTTP proxy server, which allow Web browsers in a private intranet to connect to Web servers on the Internet, but restricts all other network communications between the two networks.

pseudo-conversational. A type of CICS application design that appears to the user as a continuous conversational flow, but actually consists of multiple interconnected CICS tasks. Such a design is inherently more scalable than a conversational design, since a CICS task only executes when a user requests data, and not while the transaction is waiting for user input.

Request for Comments (RFC). A set of standards for TCP/IP and Internet protocols, managed by the Internet Engineering Task Force (IETF).

Secure Sockets Layer (SSL). A protocol designed by Netscape to enable encrypted, authenticated communications across the TCP/IP networks. It is most widely used between Web browsers and Web servers. A URL that begins with the “https” protocol indicate that an SSL connection will be used for the HTTP datastream.

servlet. A Java program that executes within a JVM on a Web server and “serves” HTML to the Web browser. Similar in concept to a CGI program.

Socket Secure (SOCKS). A proxy gateway that allows compliant client code (client code made socket secure) to establish a TCP/IP session with a remote host via means of the SOCKS gateway.

Standard Generalized Markup Language (SGML). The standard that defines several markup languages, HTML included.

System 390 (S/390). IBM’s latest mainframe hardware platform.

Systems Network Architecture (SNA). IBM’s networking protocol, that offers connections between nodes termed Physical Units (PUs), and offers programming services from Logical Units (LUs). See also LU6.2

sysplex. A set of OS/390 systems (also called MVS images or LPARs) that communicate using multi-system hardware components and software. Systems in a sysplex will share disk storage.

TCP62. A communication protocol available when using the CICS Universal Client. It utilizes the AnyNet function of IBM’s Communication Server to provide LU6.2 communication over TCP/IP (LU6.2/IP). It also provides for dynamic configuration of the client SNA node.

TCP/IP listener. In CICS, the function provided by the CICS Sockets listener task (CSOL) that handles incoming HTTP and IIOP requests.

Terminal-owning region (TOR). A CICS region that “owns” the network of attached terminals and routes the request to an attach application-owning region (AOR).

transaction. (1) In CICS, a unit of processing consisting of one or more application programs initiated by a single request. A transaction can require the initiation of one or more tasks for its execution. (2) A unit of work that transforms one or more resources from one consistent state to another consistent state. See also Logical Unit of Work.

transaction processing. A style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

transaction routing. A CICS intercommunication protocol, that enables a terminal connected to one CICS region to run a transaction in another CICS region. It is common for CICS/ESA, CICS/VSE, to have a terminal-owning region (TOR) that “owns” a network of terminals.

Transmission Control Protocol/Internet Protocol (TCP/IP). The networking protocol that forms the basis of the Internet.

URL (Uniform Resource Locator). Format used to describe the resources located on the World
Wide Web. A URL is of the form
protocol://server/pathname/document. It consists of the document's name preceded by
pathname where the document can be found, the
Internet domain name of the server that hosts the
document and the protocol (such as http or ftp)
by which the Web browser can retrieve the
document from the server.

**Virtual Telecommunications Access Method (VTAM).** Networking software for the OS/390
operating system that implements the SNA
network protocol. Now part of the IBM
Communications Server for OS/390.

**Web-aware.** A term used to denote a CICS
application that contains logic to send and
receive HTTP datastreams as opposed to 3270
datastreams.

**World Wide Web (WWW).** The collection of
resources on the Internet including HTTP
servers, newsgroups, and ftp sites.
Index

Numerics
2216 router 47
3270 Web bridge
  bridge facility 75
  garbage collection 13
  state management 13

A
APARs
  OW21511 84
  OW38086 119
  PQ31399 74
  PQ37434 96
  PQ37993 120
  PQ39719 119
  PQ49121 145
API
  DOCUMENT API 211
  WEB API 211
APPL, VTAM definition for CICS 195
application-owning region (AOR) 40, 89, 94, 128, 174, 233

B
BACKLOG, TCP/IP parameter 126
basic mapping support (BMS) 12, 15
business logic interface (BLI) 10, 13, 233
  converter 9, 233
  Decode 9, 233
  Encode 9, 233
Business Transaction Services 56

C
CBPS, autoinstall connection template 195
CICS CORBA client support 25
  See also CIOR, CIOD transactions 25
CICS dynamic routing
  distributed routing model 40
  listener region 40
CICS Native IIOP support
  See CICS CORBA client support
CICS TCP/IP listener 9, 68, 113
CICS Transaction Affinities Utility 50
CICS Universal Client 5, 15, 190

workload manager 42, 88
CICS Web Interface (CWI) 6, 233
CICS Web support 6, 55, 65, 111, 233
  See also 3270 Web bridge
  See also business logic interface
  alias 9
  See also CWBA
  analyzer 9
CICS WebServer Plugin 7, 10, 68
COMMAREA manipulation 7
HTML template manager  See DFHWBTL
Web attach transaction  See CWXN
CICSPlex SM
  Business Application Services 40
  goal algorithm 40
  queue algorithm 40
  Web User Interface (WUI) 40, 132
  Work Load Manager 39
CICSPlex SM, Web User Interface (WUI) 164
CIOD transaction 25, 95
CIOR transaction 25
classpath 187
COMMAREA 4, 10, 49, 55
Common Connector Framework (CCF) 16, 234
Common Gateway Interface (CGI) 234
common work area (CWA) 55
cookie, HTTP 53, 234
CORBA 60, 234
  CICS support of 3
  specification of 24
couple data set 84
Coupling Facility (CF) 234
Coupling Facility Data Tables 56
CPMI, mirror transaction 152
CPU constrained 66
Cryptographic Coprocessor Feature, S/390 234
CSMI, mirror transaction 10, 122
CSOL, sockets listener transaction 9, 25, 237
CTG

  applets 14
  See also Java applets
  Java class library 15
  Java methods
  See also ECI, EPI, ESI 14
  local protocol, with servlets 18, 20, 146
  network protocols 21
  servlets 14, 17, 141, 173

© Copyright IBM Corp. 2001
See also Java servlets
CWBA, alias transaction 9
CWBN, Web attach transaction 9, 126, 211

db2, data sharing 41, 56, 74, 89, 97, 107
dfh$WBRS, CWS state manager 55, 127
dfh$WBS, CWS state manager 55, 127
dfh$SSTAT, CICS statistics sample program 134
dfh$cnnv, data conversion program 9
dfh$clnt, CICS group 195
dfhcnv, data conversion table 151
dfhdyt, CICS routing program 39
dfhiiop, receiver program 25
dfhiiopa 25, 96
dfhir000, XCF group membership 84
dfhjpipe, CTG environment variable 153
dfhjpipe, environment variable 148
dfhmir, mirror program 96
dfhwba, alias program 9
dfhwpapi, CICS WebServer Plugin 7
dfhwbbl, See business logic interface
dfhwbgb, garbage collection program 13
dfhwblt, Web bridge exit 13
dfhwpca, CICS parsing program 55
dfhwbst, state management program 13
dfhwbltl, HTML template manager 8
dfhwbtt, Web terminal translation program 13
dfhxopt, EXCI options table 147
dfhxcum, EXCI URM 43, 70, 77, 84
dfhxopus, CICS CORBA URM 25
dfjiiop 25, 94
distributed program link (DPL) 4, 234
subset rules 77
dns, connection optimization 34, 67, 93, 100
document api 7
dynamic attribute of PROGRAM definition 41
dynamic DNS See DNS connection optimization
dynamic routing, in CICS 71, 85, 94
distributed routing model 39
dynamic transaction routing 107
hub routing model 39

e
definition of 234
eci
eciRequest, CTG method 88
etended logical units of work 88
enterprise access builder (EAB) 16, 234
epi 22
addTerminal, CTG method 88
CTG Java methods 15, 29
definition of 235
esi
CTG Java methods 16, 22
definition of 235
exc 5
definition of 234
pipes 43
pipes, maximum usage 70, 83
use by CICS Transaction Gateway 15
use by CICS WebServer Plugin 7, 70
EXEC CICS GETMAIN 50, 55, 127
EXEC CICS LINK 96
EXEC CICS RETURN TRANSID 50
EXEC CICS WRITEQ TS 127
EXEC CPSM GET OBJECT 148, 165
extranet 27, 235
eyuxl, CICSPlex SM routing program 39, 121

f
definition of 31
failover, test results 134, 168, 202
file-owning region (FOR) 41, 56
function shipping 235

g
go webserver api (GWAPI) 5, 7, 235

h
hierarchical file system (HFS) 235
host access class library (HACL) 28, 235
host on-demand (HOD) 26, 235
hostname, Web server directive 144
html
cookies 53, 234
definition of 235
GET 211, 216
POST 54, 212, 217
I
ICAPI
see GWAPI
ICAPI DLL, See DFHWBAPI
Interface Definition Language (IDL) 235
Internet 27, 235
internet 235
Internet Connection Secure Server (ICSS)
  See Web servers, OS/390
Internet Inter Orb Protocol (IIOP) 24, 235
intersystem communication (ISC) 236
intranet 27, 235

J
Java
  applet tag 21
  applets 27, 233
  applets, signed 21
  beans 28
  classpath 146
  Java Native Interface (JNI) 16, 23, 187, 236
  Java Virtual Machine (JVM) 17, 21, 27, 236
  servlets 17, 18, 81, 86, 164
  servlets, WebSphere AS (NT) 182
  servlets, WebSphere AS (OS/390) 145

L
ListenBackLog, Web server directive 83
listener region 40, 66, 85, 94, 112, 142, 174, 236
Local Area Network (LAN) 236
Logical Partition (LPAR) 236
loopback adaptor 177
Lotus Domino Go Webserver
  See Web servers, OS/390

M
MaxActiveThreads, Web server directive 83, 144
MAXMEMBER, couple data set 84
MaxPersistRequest, Web server directive 69
mirror transaction 122
  See also CSMI, CPMI
  private mirror 152
  multi-region operation (MRO) 236

N
netstat 132
Network Address Translation (NAT) 236

Network Dispatcher 45
  2216 router 47
  advisors 177
  cluster address 176
  configuration wizard 175
  Content Based Routing 47
  Dispatcher component 45
  HTTP Advisor 180
  Interactive Session Support 47
  loopback address 177

O
Object Management Group (OMG) 24, 236
Object Request Broker (ORB) 24
OS/390 Web server
  CICS WebServer Plugin 68
  cloning 68, 81
  configuring 144
directives
  Hostname 144
  ListenBackLog 83
  MaxActiveThreads 83, 144
  MaxPersistRequest 69
  PersistTimeout 69
  Service 10
  SSLV2Timeout 69
  SSLV3Timeout 69
  scalable mode 44, 69, 82
  server activity monitor 165

P
performance tuning 32
persistent HTTP connections 50, 69
PersistTimeout, Web server directive 69
PORT definitions, TCP/IP 143
port sharing See TCP/IP port sharing
program autoinstall, CICS 41
programming
  business logic 4
  COMMAREA manipulation 7
  presentation logic 4
  See also ECI, EPI, ESI
  Web-aware 7
  pseudo-conversations, in CICS 49

Q
queue-owning region (QOR) 56
Workload Management for Web Access to CICS

R
Request for Comments (RFC) 237
REQUESTMODEL, CICS definition 25, 95
round-robin See workload balancing

S
SDFJLOAD 97
Secure Sockets Layer (SSL) 53
S8 TCBs, in CICS 74
session-ID re-use 69
with CICS CORBA client support 97
with CWS 69
Service Level Agreement (SLA) 32
Service Location Protocol (SLP) 103
sessions, CICS definition 83, 153
shared temporary-storage 60
SHAREPORT parameter 33
SIT parameters
    AIEXIT 195
    DTRPGM 121
    EDSALIM 98
    MXT 126
    WEBDELAY 13
SOCKETCLOSE parameter 126
SOCKS 237
SOMAXCONN, TCP/IP parameter 126
SSLV3Timeout, Web server directive 69
STAT, CICS statistics transaction 134
state 31, 49
    hidden HTML form fields 54
    HTTP cookies 53
    managing with CWS 71, 127
    state tokens 51
    URL rewriting 52
    with 3270 Web bridge 75
SYNCNORRETURN 96
Sysplex Distributor 37, 67, 80, 93, 103

T
TCBs, in CICS
    QR TCB 31
    S8 74
TCP/IP listener, CICS 9
See also CSOL
TCP/IP port sharing 66, 80, 92
    configuring 120, 143
TCP/IP routing table 178

TCP62 237
TCPIPSERVICE, CICS definition 11, 66, 118
TCPIPSERVICE, closure of 137
Telnet server 100
temporary storage
    queues 50, 55
    TSMODEL definition 127
terminal control table user area (TCTUA) 50, 55
terminal-owning region 40, 107
time to live (TTL) 34, 115
Trader application
    TRADERBL 112, 121
    TRADERCV 112, 121
    VSAM files 123
TSMODEL, CICS definition 96

U
URL rewriting 52

V
virtual IP address (VIPA) 36, 67
dynamic VIPA 36
VIPA takeover 36
Virtual Private Network (VPN) 235
VSAM Record Level Sharing (RLS) 41, 56
VTAM
    APPL definition 195
    Generic Resource 38, 107
    LOGMODE 195

W
WEB API 7
Web servers
    Apache Server 17
    Domino Web server 17
    IBM HTTP Server 17, 181
    Lotus Domino Go Web server 17
    Microsoft Internet Information Server 17
    Netscape Enterprise server 17
    OS/390 Web server 10
Web-aware, CICS application 5, 7, 65, 238
WebSphere Application Server 17, 29, 144
    cloning 43
    EJB WLM 44
    OSE protocol 43
    See also OS/390 Web server
    servlet redirector 44
WebSphere Edge Server 175
workload balancing
  biased distribution 88
  definition of workload balancing 31
  round-robin 39
  workload separation 39
Workload Manager (WLM), MVS 32, 118
  compatibility mode 33
  configuring 157
  goal mode 33, 151
IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.

- Use the online Contact us review redbook form found at ibm.com/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

<table>
<thead>
<tr>
<th>Document Number</th>
<th>SG24-6118-00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redbook Title</td>
<td>Workload Management for Web Access to CICS</td>
</tr>
</tbody>
</table>

Review

What other subjects would you like to see IBM Redbooks address?

Please rate your overall satisfaction:

- O Very Good
- O Good
- O Average
- O Poor

Please identify yourself as belonging to one of the following groups:

- O Customer
- O Business Partner
- O IBM, Lotus or Tivoli Employee
- O Solution Developer
- O None of the above

Your email address:

The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.

O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.

Questions about IBM's privacy policy?

The following link explains how we protect your personal information.

ibm.com/privacy/yourprivacy/
Workload Management for Web Access to CICS
The Internet seems all-pervasive, and now it is in your CICS systems. This means, not just more work for you, but also more work for your CICS systems and your I/T infrastructure. How can you design your systems to handle it all?

In this IBM Redbook we first provide an introduction to Web-enabling CICS, followed by a brief overview of the building-blocks that provide the backbone of IBM’s workload management technologies. Included in this discussion are technologies such as DNS connection optimization, TCP/IP port sharing, and Sysplex Distributor (as provided by IBM OS/390 Communications Server); together with Network Dispatcher, CICSplex SM and CICS dynamic routing, the CICS Universal Client workload manager, and VSAM Record Level Sharing. Following this, we explain how to use these components to design a load balancing solution for whichever CICS Web-enablement strategy you have chosen.

Finally, we document three real life scenarios, where we designed, built, and tested three load balanced CICS Web-enabled solutions — a Web-aware CICS Web support application, and two servlet solutions using the CICS Transaction Gateway from both WebSphere Application Server for OS/390 and from WebSphere Application Server for Windows NT.